



## 저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

공학박사학위논문

큰 그래프 상에서의 개인화된  
페이지 랭크에 대한 빠른 계산 기법

**Fast Personalized PageRank Computation  
on Very Large Graphs**

서울대학교 대학원

전기컴퓨터공학부

박 성 찬

## Abstract

Computation of Personalized PageRank (PPR) in graphs is an important function that is widely utilized in myriad application domains such as search, recommendation, and knowledge discovery. Because the computation of PPR is an expensive process, a good number of innovative and efficient algorithms for computing PPR have been developed. However, efficient computation of PPR within very large graphs with over millions of nodes is still an open problem. Moreover, previously proposed algorithms cannot handle updates efficiently, thus, severely limiting their capability of handling dynamic graphs. In this paper, we present a fast converging algorithm that guarantees high and controlled precision. We improve the convergence rate of traditional Power Iteration method by adopting successive over-relaxation, and initial guess revision, a vector reuse strategy. The proposed method vastly improves on the traditional Power Iteration in terms of convergence rate and computation time, while retaining its simplicity and strictness. Since it can reuse the previously computed vectors for refreshing PPR vectors, its update performance is also greatly enhanced. Also, since the algorithm halts as soon as it reaches a given error threshold, we can flexibly control the trade-off between accuracy and time, a feature lacking in both sampling-based approximation methods and fully exact methods. Experiments show that the proposed algorithm is at least 20 times faster than the Power Iteration and outperforms other state-of-the-art algorithms.

**Keywords :** Graph Analysis, PageRank, Personalized PageRank, Random Walk, Power Iteration, Optimization, Algorithm

**Student Number :** 2009-30917

# Contents

<b>Abstracts</b>	<b>i</b>
<b>Contents</b>	<b>ii</b>
<b>List of Tables</b>	<b>iv</b>
<b>List of Figures</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Preliminaries: Personalized PageRank</b>	<b>4</b>
2.1 Random Walk, PageRank, and Personalized PageRank.....	5
2.1.1 Basics on Random Walk.....	5
2.1.2 PageRank.....	6
2.1.3 Personalized PageRank.....	8
2.2 Characteristics of Personalized PageRank.....	9
2.3 Applications of Personalized PageRank.....	12
2.4 Previous Work on Personalized PageRank Computation.....	17
2.4.1 Basic Algorithms.....	17
2.4.2 Enhanced Power Iteration.....	18
2.4.3 Bookmark Coloring Algorithm.....	20

2.4.4	Dynamic Programming.....	21
2.4.5	Monte-Carlo Sampling.....	22
2.4.6	Enhanced Direct Solving.....	24
2.5	Summary.....	26
<b>3</b>	<b>Personalized PageRank Computation with Initial Guess Revision</b>	<b>30</b>
3.1	Initial Guess Revision and Relaxation.....	30
3.2	Finding Optimal Weight of Successive Over Relaxation for PPR.....	34
3.3	Initial Guess Construction Algorithm for Personalized PageRank.....	36
<b>4</b>	<b>Fully Personalized PageRank Algorithm with Initial Guess Revision</b>	<b>42</b>
4.1	FPPR with IGR.....	42
4.2	Optimization.....	49
4.3	Experiments.....	52
<b>5</b>	<b>Personalized PageRank Query Processing with Initial Guess Revision</b>	<b>56</b>
5.1	PPR Query Processing with IGR.....	56
5.2	Optimization.....	64
5.3	Experiments.....	67
<b>6</b>	<b>Conclusion</b>	<b>74</b>

<b>Bibliography</b>	<b>77</b>
<b>Appendix</b>	<b>88</b>
<b>Abstract (In Korean)</b>	<b>90</b>

## **List of Tables**

1. Table of Symbols.....	4
2. Summary of PPR Computation Approaches.....	28
3. Summary of PPR Computation Algorithms.....	29
4. Selection of iteration scheme.....	33
5. Accuracy of Result in Cosine Similarity.....	72
6. Wall Clock Time for FPPR Process (Facebook Graph).....	88
7. Query Processing Time (s) for 1M Node Graphs.....	88
8. Precomputation Time (Scale-Free Network).....	89
9. Precomputation Time (Random Graph).....	89
10. Precomputation Time (Small World Graph).....	89

## List of Figures

1. Random Walk Example.....	5
2. Concept of Initial Guess Revision.....	31
3. SOR: Effect of Weight.....	35
4. Initial Guess Revision for Two Cases.....	38
5. FPPR with IGR Overview.....	43
6. Example Graph for FPPR.....	46
7. Optimization Scheme Utilizing Weighted Symmetry Property.....	50
8. Iteration Count per Node (Facebook Graph).....	52
9. Iteration Count per Node (Enron E-mail Graph).....	53
10. Computation Time per Node (Facebook Graph).....	54
11. Comparison with General Matrix Inversion.....	55
12. Effect of Utilizing Weighted Symmetry (Facebook Graph).....	55
13. Query Processing with Initial Guess Revision.....	57
14. Example System Design Utilizing Caching Scheme.....	65
15. Effect of Using Truncated Vectors.....	67



16. Effect of Re-using Previous Precomputed Data (Scale-Free Network)....	68
17. Effect of Re-using Previous Precomputed Data (Random Graph).....	69
18. Effect of Re-using Previous Precomputed Data (Small World Graph).....	69
19. Query Processing Time.....	71
20. Top-k Precision of IGR Inexact (Using Top-100 Truncated Vectors).....	73

# Chapter 1

## Introduction

Finding the proximity among multiple entities using distance/similarity measures is one of the core operations of data mining and knowledge discovery. Likewise, finding the closeness among multiple nodes within graphs is also an important problem in graph data mining. Personalized PageRank (also known as “Random Walk with Restart”) is one of most intensely studied node proximity measures for graph data mining and has also been adopted by a wide range of applications.

Personalized PageRank (PPR) is a variation of PageRank, which is a way of measuring the importance of hyperlinked webpages [32]. The core idea of PageRank is the introduction of the random walk model. It assumes that a walker resides on a node at a specific time and travels on the graph through its edges. PageRank of each node can be seen as the probability that the walker resides on each node when it infinitely “random walks” or jumps to a random node with a constant probability. Thus, PageRank can measure the importance of each node considering the link structure of graphs and it has been successfully adopted by Web search systems such as Google. Like the original PageRank, PPR is also defined using a random walk model. However, PPR assumes that a traveling walker infinitely returns (jumps) to their “restarting nodes”, which is a specific set of nodes, instead of all nodes. In PPR,

the result is skewed toward the restarting nodes, and thus, PPR is a measure of the proximity of each node to the restarting nodes. A node with a high PPR score can be considered as a node that is close to the restarting nodes. It produces relatedness scores among nodes such as the traditional distance/similarity measures, i.e., the shortest path distance and the maximal flow. PPR considers every possible direct/indirect connection among nodes while the traditional measures utilize only limited information. For example, the shortest path distance considers only the shortest connection between two nodes, while other connection information is not used. In contrast, PPR considers every path for reaching the target node that a “random walker” can follow. Thus, PPR can reflect the overall structural features of graphs.

Due to its merits, PPR has been applied to a wide range of applications such as information retrieval, context-aware recommendations, social network analysis, computational linguistics, image processing, anomaly detection, and bioinformatics.

Since PPR can be interpreted as a simple linear equation, it can be computed using basic equation solving algorithms such as iterative matrix multiplication and matrix inversion. However, for large graphs such as those representing social networks and the World Wide Web, these basic methods are not fast enough to meet the requirements of most applications. Therefore, finding efficient algorithms for PPR has been one of the major subjects of graph data processing research and, consequently, many advanced algorithms have been proposed.

Still, efficient computation of PPR within very large graphs is still an open problem. Especially previously proposed algorithms cannot handle update efficiently, thus their capability of handling dynamic graphs is severely limited. When recency is important, it can cause critical problems.

In this thesis, we present a fast algorithm for PPR computation with high and controlled precision. The proposed method vastly improves on the traditional Power Iteration in terms of convergence rate and computation time, while retaining its simplicity and strictness. The main idea of our method is to enhance power iteration method by introducing initial guess revision (IGR) and over-relaxation. By utilizing self-repairing capability of iterative method, we achieve significant performance improvement especially in terms of update handling that is important in handling large dynamic graph data such as World Wide Web. Experiments show that the proposed algorithm is at least 20 times faster than the Power Iteration and outperforms other state-of-the-art algorithms. Especially its update handling performance outperforms even state-of-the-art algorithms by orders of magnitude.

This thesis is organized as follows: In Chapter 2, we review basics on PPR. Applications and previous algorithms for PPR computation are also reviewed. In Chapter 3, we propose our main ideas to enhance PPR computation; Initial guess revision (IGR) and over relaxation. In Chapter 4, batch processing algorithm using IGR is explained, and performance gain from our idea is shown by experimental results. In Chapter 5, how IGR can contribute to PPR query processing is introduced along with its empirical evaluation is provided. Finally, we conclude the thesis in Chapter 6.

## Chapter 2

### Preliminaries: Personalized PageRank

In this section, we describe theoretical basis of PPR, including its definition and characteristics. For self-contained explanation, we start with random walk and PageRank that PPR is based on, then we provide detailed discussions on PPR based on the previously explained concepts. The difference between PageRank and PPR is also explained. Table 1 provides the definitions of the symbols that are used in the following sections.

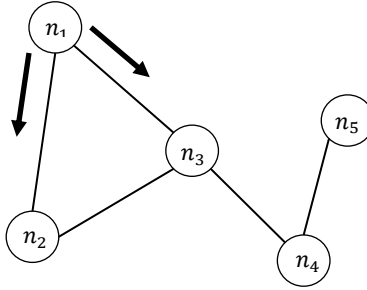
**Table 1 Table of Symbols**

Symbol	Definition
$G$	Given graph
$N$	Number of nodes in $G$
$P$	Transition matrix of $G$
$c$	Damping factor ( $1 - c$ = restart probability)
$PPR(\vec{v})$	PPR vector of a given query vector $\vec{v}$
$PPR_0(\vec{v})$	Initial guess vector for a given query vector $\vec{v}$
$n_i$	The $i^{\text{th}}$ node
$\vec{v}_i$	A unit vector defined as $\vec{v}_i[k] = \begin{cases} 1, & k = i \\ 0, & \text{otherwise} \end{cases}$
$O(n_i)$	Out neighbor node set of $n_i$
$deg(n_i)$	Degree of $n_i$

## 2.1. Random Walk, PageRank, and Personalized PageRank

### 2.1.1. Basics on Random Walk

PPR is one random walk-based proximity measure. Thus, we first start with a brief explanation of the random walk in order to provide a self-contained introduction to PPR. The *random walk* model assumes that a particle, which is called a *random walker*, walks on a given graph through its edges. For example, we can think that a random walker that resides on node  $n_1$  will be on  $n_2$  or  $n_3$  from the above assumption (Figure 1).



**Figure 1 Random Walk Example**

The state of a random walker can be expressed as a probability vector  $\vec{v}$  with  $N$  dimensions. Each dimension of  $\vec{v}$  corresponds to each node, and the entry represents the probability that a random walker resides on the node. For example, a state vector  $\vec{v}_{(0)} = (1, 0, 0, 0, 0)^T$  represents the state that a random walker resides on node  $n_1$  without uncertainty, and a state vector  $\vec{v}_{(1)} = (0, 1/2, 1/2, 0, 0)^T$  represents the state that a random walker may reside on node  $n_2$  or  $n_3$  with an even probability. We can say that if the current state is  $\vec{v}_{(0)}$ , the next state is  $\vec{v}_{(1)}$  from the assumption of the random walk. Generally, the transition of a random

walker can be represented as a probability matrix  $P^T$ , which is called the *transition matrix*. The entry of  $P^T$  can be defined by the following rule:

$$P^T_{ij} = \begin{cases} \frac{1}{|O(n_i)|}, & j \in O(n_i) \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

The transition matrix  $P^T$  for the graph in Fig. 1 can be calculated as follows:

$$P^T = \begin{pmatrix} 0 & 1/2 & 1/2 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 \\ 1/3 & 1/3 & 0 & 1/3 & 0 \\ 0 & 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \quad (2)$$

We can use the matrix  $P^T$  to compute the state vector of the next step from the state vector of the current step using the following equation:  $\overrightarrow{v_{(k+1)}} = P\overrightarrow{v_{(k)}}$ . Therefore, the state of a random walker after  $k^{\text{th}}$  steps can be expressed as  $\overrightarrow{v_{(k)}} = P^k\overrightarrow{v_{(0)}}$ , where  $\overrightarrow{v_{(0)}}$  denotes the initial state.

### 2.1.2. PageRank

Before discussing PPR, we need to review PageRank since PPR is a generalization of PageRank, a popularity measure for webpages.

PageRank assumes an imaginary Web surfer that visits one webpage at a time and randomly moves through hyperlinks. With the assumption, PageRank measures the popularity of each node by the probability of the surfer resides on each node after infinite number of steps.

This assumption can be precisely interpreted into the random walker model. Thus, PageRank can be represented by the probability distribution of the random walker over the graph.

In random walker model, the probability vector converges after infinite steps. We can compute the converged probability vector by solving the following equation:

$$\vec{r} = P\vec{r}$$

$P$  presents the  $N \times N$  transition matrix of the Markov chain that is derived from the graph. The  $i^{\text{th}}$  entry of vector  $\vec{r}$  represents the probability that the random walker resides at node  $i$ .

The probability is changed by introducing the damping factor. The justification within the random walker model is that the walker does not move over an infinite number of links but gets bored sometimes and jumps to another node at random. The following equation represents PageRank with the damping factor:

$$\vec{r} = cP\vec{r} + \frac{1}{N}(1 - c)\vec{1} \quad (3)$$

Thus,  $P\vec{r}$  represents the transition from the current state, and  $\frac{1}{N}\vec{1}$  represents the uniform state. In summary, equation (3) represents the recursive process through which a random walker follows the Markov process with probability  $c$  and jumps to random node with probability  $1 - c$ . The solution  $\vec{r}$  of equation (3) is PageRank vector, and PageRank value of a specific node  $i$  is the  $i^{\text{th}}$  entry of  $\vec{r}$ .

PageRank value represents the **centrality** of node  $i$  within the graph. The value is larger when the node is related to other important nodes. The value is smaller when the node has few edges or PageRanks of the related nodes are also small. The sum of the value is 1 because it is sum of the probabilities.



### 2.1.3. Definition of Personalized PageRank

PPR can also be represented in recursive form. The following equation represents PPR:

$$\vec{r} = cP\vec{r} + (1 - c)\vec{s} \quad (4)$$

The only difference from the original PageRank is the second term where  $\frac{1}{N}\vec{1}$  is replaced with  $\vec{s}$ . It means that now the random walker jumps to the nodes that are specified by the probability vector  $\vec{s}$ . For example, if the  $i^{\text{th}}$  entry of  $\vec{s}$  is 1 and all other entries are zeros, the random walker continuously returns to node  $i$  with probability  $1 - c$ . In PPR, the probability vector  $\vec{r}$  is relatively more skewed toward  $\vec{s}$  than in PageRank. The solution  $\vec{r}$  of the equation (4) is PPR vector, which is denoted as  $PPR(\vec{s})$ . The  $j^{\text{th}}$  entry of  $PPR(\vec{s})$  represents the **proximity** of node  $j$  from the nodes that are specified by  $\vec{s}$ . If the  $i^{\text{th}}$  entry of  $\vec{s}$  is 1 and all other entries are zeros,  $PPR(\vec{s})$  represents the proximity of each node from node  $i$ . If  $PPR(\vec{s})[j]$  is higher than  $PPR(\vec{s})[k]$ , we can say that node  $j$  is more strongly connected to node  $i$  than node  $k$  is.

In summary, the difference between PageRank and PPR is that PPR assumes that the random walker randomly returns to specific states (i.e., query states), which is unlike PageRank that assumes that the random walker returns to any node with uniform probability.

The *Personalized PageRank problem* is defined as the problem of computing  $PPR(\vec{s})$  vector when the graph and the restart vector  $\vec{s}$  are given. The *fully Personalized PageRank problem*, which is also a well-studied problem, is defined as the problem of computing the  $PPR(\vec{v}_i)$  vector for every node  $n_i$  in the given graph.

The thing that makes the Personalized PageRank problem harder than PageRank problem is that PPR has infinite possible queries. The PageRank vector remains unchanged unless the graph is changed. Thus the vector can be computed off-line and reused until the graph is updated. However,  $PPR(\vec{s})$  vectors should be recomputed when node proximity value is needed since  $\vec{s}$  can be any stochastic vector. Considering that the tasks require PPR computation such as Web search and graph data mining need frequent proximity computation, it is obvious that the algorithms for PageRank computation are not sufficient for PPR computation. Consequently, there exist tons of studies focusing on efficient PPR computation. We will provide detailed discussion on them in section 2-D.

## 2.2. Characteristics of Personalized PageRank

PPR has some useful characteristics. The most important and frequently utilized ones are the *linearity* and the *decomposition theorem* [23].

The following equation represents the *linearity*:

$$PPR(w_1\vec{u} + w_2\vec{v}) = w_1PPR(\vec{u}) + w_2PPR(\vec{v}) \quad (5)$$

It means that we can compute  $PPR(w_1\vec{u} + w_2\vec{v})$  when we know  $PPR(\vec{u})$  and  $PPR(\vec{v})$  without solving a linear equation. For example, if  $PPR((0, 0, 0, 1, 0)^T)$  and  $PPR((1, 0, 0, 0, 0)^T)$  are known, we can compute PPR vector of any linear combination of  $(0, 0, 0, 1, 0)^T$  and  $(1, 0, 0, 0, 0)^T$  such as  $(0.2, 0, 0, 0.8, 0)^T$  via weighted summation of the known PPR vectors. This property is utilized by several optimization methods such as Bookmark Coloring Algorithm.

The *decomposition theorem* is also an interesting property that is closely related to the graph structure. The following equation represents the decomposition theorem:

$$PPR(\vec{v}_1) = c\vec{v}_1 + \frac{1-c}{|O(n_1)|} \sum_{n_j \in Out(n_1)} PPR(\vec{v}_j) \quad (6)$$

It means that we can compute  $PPR(\vec{v}_1)$  if we know all PPR vectors from the out neighbor nodes of  $n_1$ . Assume that we want to compute  $PPR(\vec{v}_1)$ , and we know PPR of all out neighbor nodes. Then, in the example graph in Fig. 1, the following equation holds according to the decomposition theorem:

$$PPR(\vec{v}_1) = c\vec{v}_1 + \frac{1-c}{2}PPR(\vec{v}_2) + \frac{1-c}{2}PPR(\vec{v}_3) \quad (7)$$

Therefore, we can compute  $PPR(\vec{v}_1)$  as the summation of vectors if we know the vectors corresponding to  $PPR(\vec{v}_2)$ ,  $PPR(\vec{v}_3)$ ,  $PPR(\vec{v}_4)$  and  $PPR(\vec{v}_5)$  beforehand. Since solving linear equation is an expensive task, utilizing the above properties is one of major approaches for PPR computations.

Another important property of PPR is that it can be rewritten in summation form as follows:

$$PPR(\vec{s}) = (1-c) \sum_{k=0}^{\infty} c^k P^k \vec{s} \quad (8)$$

This interpretation of PPR can be seen as iterative summation of the state vectors for each step of a random walker that follows the transition behavior of PPR. If we assume that the initial state  $\vec{r}^{(0)} = \vec{s}$ , then the state vector of the 1<sup>st</sup> step  $\vec{r}^{(1)}$  can be computed as  $(1-c)\vec{s} + cP\vec{s}$  by equation (4). The state vector of the 2<sup>nd</sup> step can be obtained in the same fashion as follows:  $\vec{r}^{(2)} = (1-c)\vec{s} + cP\vec{r}^{(1)} = (1-c)\vec{s} + (1-c)cP\vec{s} + c^2P^2\vec{s}$ . If we repeat the former process infinitely, we can eventually reach the equation  $\vec{r}^{(\infty)} = (1-c)\vec{s} + (1-c)cP\vec{s} + (1-c)c^2P^2\vec{s} + (1-c)c^3P^3\vec{s} + \dots$ , and the equation can be rewritten as the summation form of the

equation (8). This property is utilized in several important methods including sampling-based algorithms [6][22].

Another property is *weighted symmetry*. Basically, PPR is not a symmetric measure unlike other widely used similarity measures such as Euclidean distance. That is, we cannot say that the proximity of node  $i$  from node  $j$  is equal to that of node  $j$  from node  $i$  in terms of PPR. In other words, we cannot guarantee that  $PPR(\vec{v}_i)[j] = PPR(\vec{v}_j)[i]$  always holds. However, PPR also has the property of partial symmetry when the given graph is an undirected graph. Equation (9) shows the property [25]:

$$\deg(i) \cdot PPR(\vec{v}_i)[j] = \deg(j) \cdot PPR(\vec{v}_j)[i] \quad (9)$$

Though it is not a perfect symmetric property, it can be utilized for optimization by avoiding repetitive computations. If we have the vector  $PPR(\vec{v}_j)$  beforehand, we can get the  $PPR(\vec{v}_k)[j]$  for any node  $k$  by using the property.

PPR also can be viewed as a solution of the linear system  $(I - cP)\vec{r} = (1 - c)\vec{s}$ . The system matrix  $(I - cP)$  of the above linear system has the following properties [31]:

1.  $(I - cP)$  is an M-matrix.
2.  $(I - cP)$  is nonsingular.
3. The row sums of  $(I - cP)$  are  $1 - c$ .
4.  $\|I - cP\|_\infty = 1 + c$ .
5. Since  $(I - cP)$  is an M-matrix,  $(I - cP)^{-1} \geq 0$ .
6. The row sums of  $(I - cP)^{-1}$  are  $(I - c)^{-1}$ . Therefore,  $\|(I - cP)^{-1}\|_\infty = (I - c)^{-1}$ .
7. Thus, the condition number  $\kappa_\infty(I - cP) = (1 + c)/(1 - c)$

The above properties guarantee the *existence* and *uniqueness* of the solution of the given linear system. They also assure that the solution can be computed using iterative methods for solving linear equations such as the Jacobi method.

### 2.3. Applications of Personalized PageRank

In this section, we overview the applications of PPR. As a well-studied node proximity measure, PPR has been applied to a broad range of fields including information retrieval, item recommendations, social network analysis, computational linguistics, computer vision, bioinformatics, probabilistic reasoning, etc.

#### *Personalized Web Search*

One of well-known application examples of PPR is personalized web searches. Applying PPR to personalized web searches was suggested at the birth of PageRank [32]. Its actual application was studied later by Haveliwala *et al.* [7]. In this paper, the restart vector  $\vec{s}$  is defined by topics that represent personal preferences. In particular, let  $T_j$  be the set webpages in the category  $c_j$  on the Open Directory Project<sup>1</sup>. When computing PageRank for topic  $c_j$ , the non-uniform vector  $\vec{v}_j$  is used in place of the uniform vector  $\frac{1}{N}\vec{1}$ :

$$v_j[i] = \begin{cases} \frac{1}{|T_j|}, & i \in T_j \\ 0, & i \notin T_j \end{cases} \quad (10)$$

---

<sup>1</sup> Open Directory Project, <http://odp.org/>

By using the biased restart vector  $\vec{v}_j$ , the result of PageRank can represent the importance of each page in terms of a certain topic. The result of the experiment shows that using a biased restart vector in place of an unbiased vector significantly improves the precision of an information retrieval system.

Gleich *et al.* [43] also proposed an approximate PPR algorithm and personalized web search system that can provide personalized search results to users without violating the users' privacy. Dou *et al.* [51] presented a large-scale evaluation framework for personalized search strategies and analyzed the effects of many personalization strategies.

### **Social Network Analysis**

PPR is used for measuring importance of each users in SNSs to analyze social network structures. Garcia *et al.* [55] analytically characterized all the possible values of PPR for any node, and introduced a new concept concerning the competitiveness and leadership in networks. Pedroche *et al.* [14] introduced a new parameter, the frequency, to the Leadership group, which is a group of nodes that have higher PageRanks than others. Then, they analyzed some graphs using the Leadership group while controlling the biasing factor  $\epsilon$ . Additionally, PPR has been used as a link prediction tool of SNSs [59]. Backstrom *et al.* [60] developed an algorithm called the "Supervised Random Walk" that combines the information of a network structure and edge attributes to predict the links in social network. They used the node and edge attributes to guide the random walks toward the target node. Liu *et al.* [61] proposed a link prediction method based on the local random walk that has much lower computational complexity. Xia *et al.* [44] showed that Random Walk with Restart (RWR) can be utilized to determine the relevancy in a birelational network in the bibliographic domain. Tong *et al.* [45] introduced the "center-piece

subgraphs" problem of a social network and proposed a fast subgraph extraction algorithm. Jung *et al.* [62] proposed Signed Random Walk with Restart (SRWR) for personalized rankings in signed networks using a signed surfer. Devooght *et al.* [63] introduced a random walk based modularity measure, which is computed using the paths instead of the traditionally used edges, for analyzing social networks.

### **Computational Linguistics**

PPR has made some achievements in the field of Computational linguistics. Liu *et al.* [16] applied PPR to generate query-based multidocument summarizations. They used PPR to rank the personalized prior probability of each sentence, which is computed using their salience model and relevance model. Agirre *et al.* [15] used PPR to solve the graph-based word sense disambiguation (WSD) problem. In WSD, a graph consists of nodes, which represent word senses, and edges, which represent relations between pairs of word senses. PPR is utilized when performing disambiguation by applying a ranking algorithm to a graph. Pershina *et al.* [64] introduced PPR-based random walk method to solve Named Entity Disambiguation (NED) problem. They used PPR algorithm on a graph where the vertices represent candidate links and the edges represent links in Wikipedia. They achieved state-of-the-art performance on a 27.8K named entity mention dataset.

### **Computer vision**

Computer vision is one of the fields where PPR has also been widely applied. Kim *et al.* [19] addressed a multilabel supervised image segmentation problem when initial labels of some pixels are given. They introduced the generative model for image segmentation using the steady-state probability of RWR.

Since RWR considers all relevance relations between the nodes in a graph (image), it is effective at addressing the texture problem. Ham *et al.* [20] proposed a generalized random walk with restart (GRWR), which is a generalized version of RWR that adopts local and nonlocal approaches for image regularization. They applied GRWR to depth map upsampling and interactive image segmentation and showed that the GRWR is more robust to outliers and can aggregate texture information better. Wang *et al.* [65] utilized PPR to refine image annotations. After a relevance model-based algorithm determines the candidate annotations, RWR is used to rerank the annotations based on the corpus information and original confidence. Kim *et al.* [66] proposed a multiscale saliency detection algorithm that uses RWR to refine a saliency map. Similarly, Kim *et al.* [67] proposed a spatiotemporal saliency detection algorithm for video sequences based on RWR. Lee *et al.* [68] proposed a robust dense stereo reconstruction algorithm using RWR. Kim *et al.* [69] devised a modified data-driven RWR framework that can incorporate locally adaptive and data-driven restarting probabilities to handle the colorization problem of grayscale images. Oh *et al.* [70] presented a probabilistic method for correspondence matching using RWR.

## **Bioinformatics**

PPR can also be applied to scientific data analysis [21][47][48]. Iván *et al.* analyzed protein interaction networks using PPR [21]. They applied PPR to analyze protein-protein interaction (PPI) networks that connect interacting proteins. Sun *et al.* [71] proposed a global network-based computational framework, which was called RWRlncD, to infer potential human lncRNA-disease associations by implementing the method on an lncRNA functional similarity network. Chen *et al.* [72] proposed an Improved Random Walk with Restart for a lncRNA-Disease Association



Prediction (IRWRLDA) model to predict novel lncRNA-disease associations by incorporating lncRNA expression similarity and disease semantic similarity. Chen *et al.* [73] proposed a Network-based Random Walk with Restart on a Heterogeneous network (NRWRH) to predict potential drug-target interactions on a large scale under the hypothesis that similar drugs often target similar target proteins. Li *et al.* [74] identified novel epigenetic factors by using a computational method that applied RWR algorithm on a protein-protein interaction (PPI) network using reported epigenetic factors as seed nodes. Blatti *et al.* [75] presented a network-based method, which involves RWR, for ranking the genes or properties related to a given gene set. Chipman *et al.* [76] presented a method based on RWR, which captures aspects of the network topology to classify potential genetic interactions and applied it to biological networks.

## **Others**

There are many other fields where PPR is applied. The FolkRank is a folksonomy-based algorithm that is used for tag recommendations. Kim *et al.* [17] proposed a new way to efficiently compute the FolkRank by representing it as a linear combination of PPR vectors.

In first-order probabilistic representation systems, inference by grounding can be very computationally expensive. Wang *et al.* [42] proposed a first-order probabilistic language to approximate the local grounding by applying PPR to a small graph.

Nykl *et al.* [77] evaluated a citation network that was built using the ISI Web of Science database. Their aim was to find an evaluation method that best matches the list of authors who received ACM Fellowships or ACM SIGs. The best ranking method included PPR where the personalization is based on PageRank journal values.

Local graph diffusion is an effective tool for solving graph clustering problems. Avron *et al.* [78] proposed an efficient local algorithm for approximating a graph diffusion, which generalizes PPR and the heat kernel.

Tabrizi *et al.* [52] proposed a Personalized PageRank Clustering (PPC) algorithm that utilizes the random walk and modularity to accurately reveal the inherent clusters of graphs. It also gives a hierarchy of the clusters given linear time and space complexity.

Andersen *et al.* [50] presented a generalized local partitioning algorithm for undirected graphs to strongly connect directed graphs by computing PPR vector.

Guo *et al.* [13] proposed the Access Time-length and Frequency-based PageRank to prefetch web pages for web page caching.

With respect to databases, Balmin *et al.* [38] devised a method called "ObjectRank", which is a variation of PPR, to perform keyword searches based on authority. Chakrabarti [40] proposed HubRank, which is a proximity search platform for Entity-Relation graphs, using the dynamic PPR.

## 2.4. Previous Work for Personalized PageRank Computation

### 2.4.1. Basic algorithms

There are two basic algorithms to solve the problem: power iteration and direct solving. These algorithms are directly derived from the definition of Personalized PageRank. Power iteration is doing the following iteration until  $\vec{r}^{(n)}$  is converged:

$$\vec{r}^{(n+1)} = cP\vec{r}^{(n)} + (1 - c)\vec{s}$$

Direct solving method solves the following equation:  $\vec{r} = cP\vec{r} + (1 - c)\vec{s}$ . Thus,

solution

$$\vec{r} = (1 - c)(I - cP)^{-1}\vec{s}$$

is the answer to the Personalized PageRank query about the query state  $\vec{s}$ . These algorithms have obvious drawbacks. First, power iteration algorithm requires multiple times of matrix-vector multiplications. One iteration requires  $O(N^2)$  time. Direct solving algorithm is even worse. As we can see in the equation, it includes computation of inverse matrix. Since its time complexity is  $O(N^3)$ , it can be applied to small sized graph data only. Furthermore, generally matrix inversion does not preserve the sparsity of the original matrix. It means we cannot utilize the sparse matrix representation of graphs. From the observation, we can see that the basic algorithms cannot handle large-scale graphs. To solve the situation, there have been large numbers of studies about efficient computation of Personalized PageRank.

#### 2.4.2. Enhanced power iteration

These studies can be viewed as advanced versions of the power iteration. Actually, the power iteration itself is identical to the *Jacobi method*, which is the most basic iterative method of solving general linear equations. Since equation (4) is a linear equation, any iterative method for linear equation solving can be applied, and as we stated in section 2.D, PPR equation converges to the unique solution with iterative methods due to its mathematical properties. Therefore, linear equation solving algorithms such as the Gauss-Seidel method, Successive Overrelaxation (SOR), GMRES, and Multigrid methods can be applied to the problem. Though these iterative methods cannot produce exact solutions to the problems, their errors can be

controlled and the error bound can be strictly defined unlike other approximation algorithms such as sampling-based methods. On the top of applying those general equation solvers, diverse techniques are devised for the specific problem of computing PPR.

One of the most recent studies in this category utilizes the GMRES [10]. They utilize the GMRES for solving the equation (4), and applied a *preconditioning* method to accelerate the convergence of iterative algorithm. General concept of preconditioning is solving  $MA\vec{x} = M\vec{b}$  to solve  $A\vec{x} = \vec{b}$ . The matrix M is called the *preconditioner*, and a well-chosen preconditioner reduces the number of iterations. It is known that the matrices that are close to  $A^{-1}$  can be good preconditioners. To construct a good preconditioner, they apply core-tree decomposition. They use an inversion of the tree-like part of the original graph as a preconditioner. The following table shows the result of the performance evaluation.

Overall, their method performs better than the power iteration and naïve GMRES. They report that their algorithm reduces the iteration count to achieve the same accuracy by 1/5 comparing to power iteration and 1/3 to naïve GMRES. However, their algorithm still requires several minutes to process each query on large graphs with millions of nodes.

Additionally, there are several studies seeking to accelerate the iterative method by tuning the power iteration. The extrapolation method is an example of the approach [30]. They accelerate the power iteration by subtracting nonprincipal eigenvectors periodically. They report that the extrapolation method can reduce the number of iterations of the power iteration by 1/2. It is an effective methods to solve the problem, but it is developed mainly for PageRank problem and its scalability for PPR

computation is not guaranteed.

### 2.4.3. Bookmark Coloring Algorithm

Bookmark Coloring algorithm (BCA) [6] is also an iterative algorithm similar to the power iteration. However, BCA asynchronously updates PPR vectors while the power iteration does it in a synchronous manner. BCA utilize equation (8) in section 2.D. With the interpretation, PPR can be computed in a cumulative way. The basic BCA for computing  $PPR(\vec{v}_i)$  can be interpreted using a recursive function  $BC(j, w, c)$ .

1. Set  $\vec{p} \leftarrow c\vec{v}_i$  when  $j = i$ ,  $\vec{p} \leftarrow \vec{0}$  otherwise.
2. If stopping criterion is met, return  $\vec{p}$ .
3. For all  $n_k \in O(n_j)$ , do  $\vec{p} \leftarrow \vec{p} + BC(k, (1 - c)w/\deg(n_k), c)$ .
4. Return  $\vec{p}$ .

Though the above algorithm is defined as a recursive function, BCA can also be implemented using a queue data structure [6]. This scheme provides a more efficient implementation utilizing sparsity than matrix-based algorithms such as the power iteration. This version of BCA is also called ForwardPush algorithm [84]. It is possible to compute  $PPR(\vec{v}_i)[j]$  for the given target node  $j$  and every source node  $i$  by reversing the weight propagation direction from the above algorithm. The reversed one is called ReversePush algorithm [84]. Several recent studies combine the Push scheme and Monte-Carlo sampling to improve query processing time.

BCA gradually completes the summation that is defined in equation (8) using repeated asynchronous updates. It can also be viewed as an asynchronous version of the power iteration, and its convergence rate is also identical to the power iteration. That means that BCA requires large numbers of iterations since the basic algorithm cannot benefit from advanced algorithms such as the GMRES in terms of the number of iterations.

If some  $PPR(\vec{v}_h)$ s are known, BCA can be computed in a more efficient way utilizing the decomposition theorem. We stop the recursive calls and directly construct the return value using known vectors. HubRank [5] is a revised version of BCA with a smart hub selection algorithm and approximation. It provides better performance in terms of the query processing time; however, it still requires a long precomputation time. In an empirical evaluation, it takes over 20 hours to compute PPR vectors for the selected hubs to achieve easonable query response time.

#### 2.4.4. Dynamic programming

Dynamic programming method for Personalized PageRank utilizes the decomposition theorem [23]. The most basic algorithm is to repeat update by decomposition theorem until Personalized PageRank vectors are converged.

1. Set  $PPR^{(0)}(\vec{v}_i) \leftarrow c\vec{v}_i$  for all  $n_i$ .
2. Update  $PPR^{(k+1)}(\vec{v}_i) \leftarrow c\vec{v}_i + \frac{1-c}{|O(n_i)|} \sum_{n_j \in O(n_i)} PPR^{(k)}(\vec{v}_j)$  for all  $n_i$ .

3. Repeat the step 2 until converges.

Like bookmark coloring algorithm, dynamic programming also updates vectors by repetitively applying decomposition theorem. However, unlike bookmark coloring algorithm, dynamic programming updates every  $PPR(\vec{v}_i)$  simultaneously. This algorithm can be applied to Fully Personalized PageRank problem since it produces  $PPR(\vec{v}_i)$  from every node simultaneously; however, it requires large space to store intermediate results. To reduce space requirement, two methods are applied: rounding and sketching [12]. Rounding technique optimize its space requirement by rounding all values down to a multiple of the prescribed error value  $\epsilon$ . The second one, sketching is hash based approach originally proposed for stream data processing. It reduces space requirement dramatically; however, it takes far longer and does not guarantee error bound.

#### **2.4.5. Monte-Carlo sampling**

Monte-Carlo sampling is one traditional method to solve problems with high complexity. It avoids the complexity by deriving the solutions using samples that are produced with a large number of trials. This method can be easily applied to the Personalized PageRank problem. In this case, the sample database is a collection of random walk traces, that is, a set of node sequences (which are called “fingerprints”).

There are two Monte-Carlo methods for the Personalized PageRank problem: the MC end-point and the MC complete path [22]. The MC end-point method uses the

summation form (equation (8)). If we know  $P^k \vec{s}$  with enough depth  $k$ ,

We can compute  $PPR(\vec{s})$  using simple algebraic operations, and  $P^k \vec{s}$  can be approximated by counting the end points of the sample traces with length  $k$  from the starting node that is specified within  $\vec{s}$ . While the MC end-point uses only the end point of each trace, the MC complete path method uses every node within traces.

Monte-Carlo scheme is mainly utilized in the fully Personalized PageRank problem since it does not requires a square space to store all computed PPR vectors. One of most important algorithms in this category is doubling [4]. It populates long traces by concatenating short traces, and the procedure is defined using MapReduce framework. In the empirical evaluation, the doubling algorithm performs approximately 8 times faster than rounding (section 4.E). In spite of its impressive enhancement, it does not strictly guarantee the error bound because of the substantial limitation of sampling method, and it is hard to say that it obviously outperforms other advanced approaches considering that rounding is a relatively underperforming algorithm that has an identical convergence rate to the power iteration.

One of recent studies, FAST-PPR, also adopts the Monte-Carlo scheme. It solves point-to-point Personalized PageRank problem that computes  $PPR(\vec{v}_i)[j]$  when the graph and two nodes (source and sink) are given [11]. It utilizes two sided random walk samples (from the source and sink) at the same time. It can be viewed as the combination of Monte-Carlo sampling and ReversePush that is discussed in section 4.C. As a result, it achieves impressive performance. However, it is unclear how to utilize the point-to-point PPR in the applications since the entries of PPR vectors are relative values. Another recently proposed method, PowerWalk [88] is also a sampling-based algorithm. To produce accurate answers, it processes queries in



iterative manners based on the sampled information.

#### 2.4.6. Enhanced direct solving

The direct equation solving method that solves linear equations by computing the inversions is not generally recommended due to its high complexity. However, when approximated answers are acceptable, revised versions of the method that produce approximated solutions can be an alternative method to solve PPR problems.

One of pioneering studies in this category introduces the low rank approximation to reduce the inversion complexity [1]. They use singular value decomposition (SVD) for the low rank approximation. By using the Low rank approximation and Sherman-Morrison lemma [79], it computes PPR using a smaller inversion matrix. It can be achieved as follows. When  $\tilde{P} = USV$  and  $\tilde{A} = (S^{-1} - cVU)^{-1}$ , then  $(I - cP)^{-1} \cong (I - cU\tilde{A}V)$  holds according to the Sherman-Morrison lemma. Since the dimension of  $S$  is far smaller than  $P$ ,  $(I - cU\tilde{A}V)$  can be computed more efficiently than  $(I - cP)^{-1}$ . This method provides a good approximation for PPR problem; however, its error bound is hard to control. They reported that their algorithm performs better than direct inversion and the power iteration. However, they use a relatively small graph (with 315K nodes) in the experiment.

There is also a study that optimizes direct equation solving without introducing the approximation [9]. They reorder the dimensions of the overall equations before the inversion to preserve the sparsity of the original matrix.

Generally, matrix inversion can be computed via LU decomposition, and the result

is computed as  $U^{-1}L^{-1}$ . With the proper reordering, the sparsity of the original matrix is also preserved with respect to  $U^{-1}$  and  $L^{-1}$ , and the sparse matrices can be stored and processed more efficiently than dense matrices.

Though it can efficiently process queries (one matrix-vector multiplication per each query), it requires massive precomputation time since it performs LU decomposition that requires over  $O(N^2)$  time. Considering that the size of the datasets used in their experiments are relatively small (under 300K nodes), it is hard to say whether it can work with very large graphs with over 10M nodes.

BEAR (Block Elimination Approach for Random Walk with Restart on Large Graphs) is one of important recent advancement [27]. It basically follows the studies of the optimized direct equation solving category according to our taxonomy. BEAR reduced the dimension of the matrix that is to be inverted by the block elimination using the Schur complement method [28]. When using the Schur complement method, we need to invert only some submatrices and the Schur complement matrix, which is smaller than the whole matrix.

For an effective application of the Schur complement method, the adjacency matrix should have a large and easy-to-invert submatrix such as a block diagonal matrix. The BEAR utilizes reordering and clustering to easily invert a system matrix, similar to in Fujiwara *et al.* [9]. BEAR decomposed the graph into hubs and spokes. Within each connected component containing spokes, BEAR reorders the nodes in ascending order of the degrees within the component. As a result, BEAR can get an adjacency matrix whose upper-left area is a large and sparse block diagonal matrix that is easily inverted, while the lower-right area is a small but dense matrix.

In the rest of the preprocessing step, BEAR precomputes several matrices including the Schur complement. To solve the equation using the block elimination method, it requires inverting the Schur complement matrix and block diagonal submatrix. BEAR inverts these matrices using LU decomposition. In the query phase, the BEAR quickly computes PPR scores for a given query node using the matrices that are computed in the preprocessing step. The BEAR takes less time and memory space than other preprocessing methods. Their experiments show that their algorithm runs 300 times faster than the simple iterative method.

## **2.5. Summary**

In the previous section, we review the major studies on the computation of PPR. Each approach has distinctive characteristics, and one should choose the most appropriate algorithm by considering the most suitable characteristics to properly utilize PPR. Table 2 summarizes the characteristics of the five approaches that presented in the previous section.

In terms of the precision, direct equation solving is the best algorithm. If errors are totally unacceptable, direct equation solving is the only valid solution. Though their precomputation costs are high, the advanced techniques based on approaches such as the BEAR reduce the large amount of precomputation time. Iterative equation solving, the Bookmark Coloring algorithm, and Dynamic Programming cannot produce exact answers; however, their errors can be controlled since they guarantee a predefined error bound. With a very small error bound, the outputs can be viewed as near exact answers. One should consider using approaches other than exact

algorithms since a large portion of the applications beside scientific data analysis do not require exact computations of PPR.

When an application requires short precomputation time, direct equation solving-based techniques and Monte-Carlo sampling can be considered as improper solutions. In contrast, iterative methods and Bookmark Coloring do not require precomputation phases. Though iterative methods require longer query time computations, this can be overcome by adopting an advanced iterative scheme such as preconditioning and overrelaxation. Some advanced techniques based on approaches such as HubRank reduce the query time computations by introducing precomputation phases. One should consider these techniques when the query time computation is problematic.

The auxiliary data size also should be considered. Direct equation solving requires space to store the matrix inversion, and Monte-Carlo sampling must store large numbers of random walk traces to achieve high precision while iterative methods and Bookmark Coloring require no auxiliary data space. If no additional storage to support PPR computation is available, the most basic iterative methods and Bookmark Coloring algorithms should be introduced. However, there exist several advanced techniques that utilize additional space to reduce the computation time. If the situation is not extremely limited in terms of space, one should consider those advanced algorithms.

The common problem of recent algorithm is long precomputation time. Recently proposed algorithms such as BEAR require precomputation phase to optimize query answering performance. However, long precomputation time limits their capability of dealing with dynamic graphs thus their application can be limited too. Thus, overcoming this high precomputation cost is one of main research subject.

**Table 2 Summary of PPR Computation Approaches**

	<b>Pre-computation Time</b>	<b>Query Answering Time</b>	<b>Auxiliary Data Size</b>	<b>Precision</b>
<b>Iterative Equation Solving</b>	None	High	None	Guaranteed Error Bound
<b>Direct Equation Solving</b>	Very High	Low	Large	Exact
<b>Bookmark Coloring</b>	None	High	None	Guaranteed Error Bound
<b>Dynamic Programming</b>	Very High	Low	Large	Guaranteed Error Bound
<b>Monte-Carlo Sampling</b>	High	Low	Large	Probabilistic Error Bound

**Table 3 Summary of PPR Computation Algorithms**

Category	Publication info	Feature	Auxiliary Data	Precision
<b>Direct Equation Solving</b>	ICDM 2006 [1]	Low rank approximation	Approximated inverse matrix	Inexact
	VLDB 2011 [9]	Matrix inversion after reordering	Decomposed inverse matrix	Exact
	SIGIR 2013 [83]	Updating precomputed PPR vectors in direct equation solving scheme	Not required	Exact
	SIGMOD 2015 [27]	Matrix inversion after block elimination	Decomposed inverse matrix	Exact
	SIGMOD 2017 [58]	Matrix inversion after block elimination, and fine tuning the result with iterative methods	Decomposed inverse matrix	Inexact result with guaranteed error bound
<b>Iterative Equation Solving</b>	WWW 2003 [30]	Accelerated power iteration via extrapolation.	Not required	Inexact result with guaranteed error bound
	KDD 2010 [25]	Turning the nodes with high out-degree into sinks to reduce computation cost.	Not required	Inexact
	VLDB 2014 [10]	Iterative method with preconditioning by core-tree decomposition	Result of core-tree decomposition	Inexact result with guaranteed error bound
	KDD 2016 [84]	A variant of power iteration for dynamic graphs	Not required	Inexact
	WWW 2018 [85]	A variant of power iteration for dynamic graphs	Not required	Inexact result with guaranteed error bound
<b>Bookmark Coloring Algorithm</b>	Internet Math. 2006 [6]	Basic bookmark coloring algorithm (BCA)	Not required	Inexact result with guaranteed error bound
	VLDB Journal 2011, WWW 2007 [5]	Revised BCA using precomputed PPR vectors of selected nodes	Precomputed PPR vectors of selected query nodes	Inexact
	KDD 2015 [86]	Updating precomputed PPR vectors by BCA-like procedure	Not required	Inexact result with guaranteed error bound
<b>Dynamic Programming</b>	WWW 2003 [23]	Basic dynamic programming for PPR	Precomputed PPR vectors of selected query nodes	Inexact result with guaranteed error bound
	WWW 2006 [12]	Dynamic programming with rounding & sketch	All PPR vectors need to be stored while processing	Inexact
<b>Monte-Carlo Sampling</b>	Internet Math. 2005 [22]	Basic Monte-Carlo sampling for PPR	Sampled random walk traces	Inexact
	SIGMOD 2011 [4]	Monte-Carlo sampling with doubling	Sampled random walk traces	Inexact
	KDD 2014 [11]	Combining Monte-Carlo sampling and ReversePush	Not required	Inexact
	CIKM 2016 [88]	Sampling-based method with iterative query processing	Sampled random walk traces	Inexact
<b>Other</b>	VLDB 2011 [8]	Top-k query processing by error bound estimation	Not required	Exact top-k list
	ICML 2014 [26]	Anti-differentiating approximation	Not required	Exact
	VLDB Journal 2015 [29]	Scheduling with hub info	Tours and reachability information	Inexact
	WSDM 2016 [80]	Top-k query processing by error bound estimation, and combination of Monte-Carlo sampling and ReversePush	Sampled random walk traces	Inexact top-k list
	VLDB 2016 [81]	Top-k query processing by error bound estimation, and combination of Monte-Carlo sampling and ReversePush	Sampled random walk traces	Inexact top-k list
	KDD 2017 [82]	Top-k query processing by error bound estimation, and combination of Monte-Carlo sampling and ForwardPush	Sampled random walk traces	Inexact top-k list

## Chapter 3

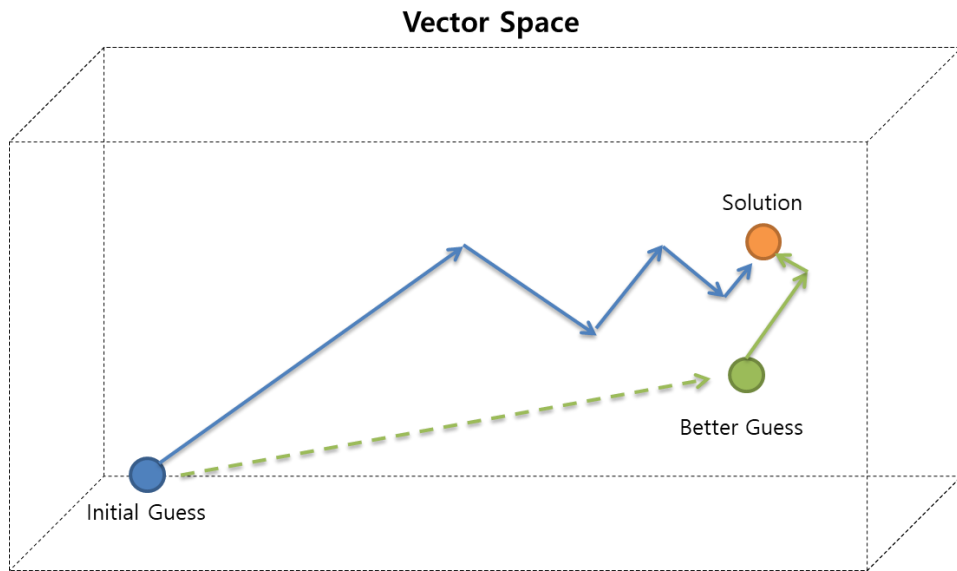
### Personalized PageRank with Initial Guess Revision

In this chapter, we explain a novel PPR computation algorithm using initial guess revision and relaxation. As we discussed in the former section, long precomputation time is one of major drawbacks of current PPR computation algorithms, and even state-of-the-art algorithms require heavy precomputation to achieve efficient query answering. To cope with the issue, we utilize the iterative method that has got relatively less attention in recent studies. Though it has a drawback that it requires repetitive matrix-vector multiplications, it has an important characteristic that can be utilized to solve the issue: it is **self-repairing** capability. Iterative method can find the correct solutions by refining half-baked or wrong solutions while other methods such as matrix decomposition-based methods cannot utilize them. This capability brings out significant performance gain especially in terms of update handling. Thus, our idea is summarized as two aspects: how to accelerate the iterative process itself, and how to utilize the former known vectors efficiently. The first one is **relaxation** and the last one is **initial guess revision**. We devised specialized method for PPR computation utilizing mathematical properties of PPR on the top of those two ideas. The discussion on those two ideas is provided throughout this chapter.

### 3.1. Initial Guess Revision and Relaxation

The basic idea is to enhance iterative method (power iteration method), and we enhance it by introducing 1) Initial Guess Revision and 2) Relaxation method into iterative computation for PPR.

Firstly, initial guess revision (IGR) boosts convergence of iterative method by using better initial points for iteration, and the initial points are constructed by re-using known PPR vectors. The figure 2 illustrates how initial guess revision contributes to faster computation of PPR.



**Figure 2 Concept of Initial Guess Revision**

Most iterative equation solving algorithms start with some initial points, and without clues, default initial points such as zero vector are used. However, if we have some clues and we can guess the initial points that are closer to the target solutions, the



iteration count to reach the solution can be dramatically reduced. With better guess, we can find the solution faster. However, good initial guesses are hard to find for general linear equation solving, thus the importance of choosing good initial guesses is often overlooked. However, PPR has some useful mathematical properties such as *decomposition theorem* and *linearity* as we discussed in chapter 2.2, and we devise an algorithm for guessing good initial guesses for PPR problem. The detail algorithm is discussed in chapter 3.3.

We can further boost convergence rate by applying *relaxation* scheme on the top of initial guess revision. Relaxation is an acceleration method for iterative linear equation solving defined as follows:

$$x^{(n+1)} \leftarrow (1 - w)x^{(n)} + wG(x^{(n)})$$

$G(x^{(n)})$  is the update result from other (simpler) method such as power iteration. We utilize *over-relaxation* scheme that boosts convergence by giving more weight to the updated result by setting  $w$  larger than 1.0.

One may question that why the relaxation method is selected for PPR computation instead of more recently proposed methods such as Krylov subspace method and multigrid method. However, they are not suitable for large scale parallel implementation that is required for large scale PPR computation. Also, they are mainly devised for the problems with relatively low dimension (1K ~ 10K). Using basic textbook methods such as Jacobi method and Gauss-Seidel method instead is not proper solution too since their convergence rate is intolerably slow for PPR

computation. Thus, introducing relaxation is most adequate method for solving PPR. On the top of the feature, it is known that Krylov subspace methods cannot be accelerated by the choice of initial points. Thus, it is reasonable to fuse the initial guess revision and relaxation method for optimizing PPR computation instead of Krylov subspace methods such as Conjugate Gradient and GMRES. Multigrid methods are not chosen for the same reason. They are not suitable for massive equation solving with over millions of dimensions.

**Table 4 Selection of iteration scheme**

Method	High convergence rate	Parallelism friendly
Jacobi method	X	O
Gauss-Seidel method	X	O
<b>Relaxation method</b>	<b>O</b>	<b>O</b>
Krylov subspace methods	O	X
Multigrid method	O	X

With relaxation scheme, the iterative update process for PPR is changed as follows:

$$PPR^{(n+1)}(\vec{v}_l) \leftarrow (1 - w)PPR^{(n)}(\vec{v}_l) + w(cP \cdot PPR^{(n)}(\vec{v}_l) + (1 - c)\vec{s})$$

Especially we use successive over-relaxation (SOR) for PPR computation that is a variant of relaxation method. Since we use over relaxation, the term  $PPR^{(n)}(\vec{v}_l)$  has negative weight. Though the update process is expressed in more complex formula than power iteration, it requires only one weighted summation of two

vectors in addition to the original power iteration. However, the performance gain from introducing over relaxation is not trivial. That is, power iteration can be boosted by nearly negligible additional computation. The empirical evaluations on the strategy is presented in Chapter 4.3.

### 3.2. Finding Optimal Weight of Successive Over Relaxation for PPR

Generally, choice of optimal weight  $w$  of SOR is not a trivial problem since it requires finding spectral radius of system matrices in linear equation. However, in PPR problem, we can find the optimal weight by simple algebraic calculation utilizing the fact that the transition matrix  $P$  is a *stochastic matrix*, and spectral radius of stochastic matrices is 1. Thus, with the assumption that there is no cycle, no island node, and no dangling node (hence the transition matrix  $P$  is a proper stochastic matrix), we can derive the optimal weight as follows:

A. Generally,  $w_{opt} = 1 + (\frac{\mu}{1+\sqrt{1-\mu^2}})^2$  if  $\mu = \rho(C_{jac})$

B.  $C_{jac} = cP$  by the following facts:

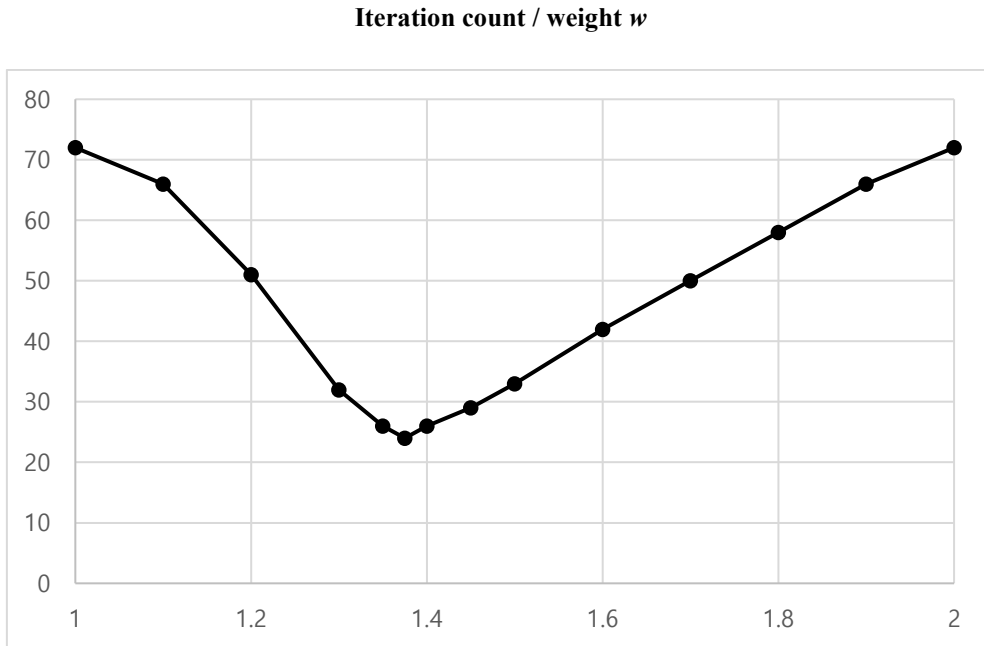
i.  $C_{jac} = I - D^{-1}A$ , and

ii.  $D = I$ ,  $A = (I - cP)$  for PPR equation

C.  $\rho(cP) = c$  since  $P$  is a stochastic matrix

D. Thus,  $w_{opt} = 1 + (\frac{c}{1+\sqrt{1-c^2}})^2$  for PPR equation

In summary,  $1 + (\frac{c}{1+\sqrt{1-c^2}})^2$  is the optimal weight we want to find. The notable feature of the formula is that it only depends on the damping factor  $c$ . The transition matrix  $P$  and the query vector  $\vec{s}$  do not appear in the formula. That is, we do not need to consider the structural characteristics of the given graph and the query vector to determine the optimal weight. This is a largely helpful property that we do not need to analyze the structure of graphs to find the optimal weights. However, in real graphs, the assumption that  $P$  is a proper stochastic matrix is not always true. Thus, technically speaking, the  $w_{opt}$  derived by the above formula is only an estimation value for the real optimal weight. However, it still provides a good starting point to find the real optimal points.



**Figure 3 SOR: Effect of Weight**

The figure 3 shows the iteration count by the choice of weight. We can see that it has simple curve that has only one and unique optimal point. It is also a useful property for finding the real optimal weight from the theoretic optimal weight  $w_{opt} = 1 + (\frac{c}{1+\sqrt{1-c^2}})^2$ .

### 3.3. Initial Guess Construction Algorithm for Personalized PageRank

In this section, we discuss the algorithm to construct initial guesses for PPR. Generally, good initial guess is hard to obtain, and plain starting points such as a zero vector or random vectors are used instead. However, PPR has several useful mathematical properties. We explain how to construct good initial guesses using the properties in the following section.

As we stated before, we utilize decomposition theorem for constructing good initial guesses. Decomposition theorem can be used to deduce  $PPR(\vec{v}_i)$  if we know  $PPR(\vec{v}_j)$  for every neighbor node  $n_j$  of  $n_i$ . In the case, initial point guessing is not required since we can compute the exact vector  $PPR(\vec{v}_i)$  without performing iterations. Otherwise, iterative update process is required, and iteration counts can be largely reduced by using better initial guesses instead of zero vector or other default initial points such as  $c\vec{v}_i$ .

In summary, there are two cases:

- Compute  $PPR(\vec{v}_1)$  directly by decomposition theorem
- Create initial guess  $PPR_0(\vec{v}_1)$  by initial guess construction algorithm

In the first case,  $PPR(\vec{v_1})$  is directly derivable. On the other hand, we need to compute the initial guess  $PPR_0(\vec{v_1})$  in the second case, and it can be done by the following algorithm:

1. Set  $PPR_0(\vec{v_{nid}}) \leftarrow c\vec{v_{nid}}$
2. For each neighbor node  $neid$  already computed

$$PPR_0(\vec{v_{nid}}) \leftarrow PPR_0(\vec{v_{nid}}) + \frac{1-c}{d} PPR(\vec{v_{neid}})$$

3. For each neighbor node  $neid$  not computed

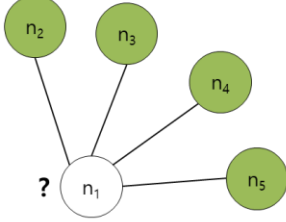
$$PPR_0(\vec{v_{nid}}) \leftarrow PPR_0(\vec{v_{nid}}) + \frac{c(1-c)}{d} \vec{v_{neid}}$$

The overall process is similar to computing  $PPR(\vec{v_1})$  by decomposition theorem. It adds  $\frac{1-c}{d} PPR(\vec{v_{neid}})$  vectors for each neighbor node  $n_{neid}$  to the default guess  $c\vec{v_{nid}}$ . If  $PPR(\vec{v_{neid}})$  is unknown,  $c\vec{v_{neid}}$ , the default guess for  $PPR(\vec{v_{neid}})$  is used instead. In this way, we can acquire half-solved vectors for PPR problem, and they can be utilized as better initial guesses.

The figure 4 illustrates how the initial guess construction algorithm works in each case. In the case #1, the PPR vector  $PPR(\vec{v_{neid}})$  for each neighbor node is already known. Thus,  $PPR(\vec{v_1})$  can be acquired simply applying decomposition theorem. However, in the case #2,  $PPR(\vec{v_2})$  and  $PPR(\vec{v_4})$  are unknown, hence the

**CASE #1:**

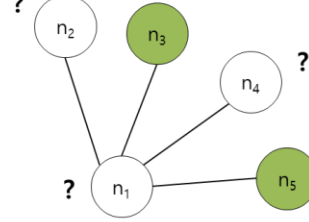
Compute  $PPR(\vec{v}_1)$  directly by decomposition theorem



$$\begin{aligned} PPR(\vec{v}_1) &= c\vec{v}_1 + \frac{1-c}{4} PPR(\vec{v}_2) + \frac{1-c}{4} PPR(\vec{v}_3) \\ &+ \frac{1-c}{4} PPR(\vec{v}_4) + \frac{1-c}{4} PPR(\vec{v}_5) \end{aligned}$$

**CASE #2:**

Create initial guess  $PPR_0(\vec{v}_1)$  and run iterative solver from the guess



$$\begin{aligned} PPR_0(\vec{v}_1) &= c\vec{v}_1 + \frac{1-c}{4} PPR(\vec{v}_3) + \frac{1-c}{4} PPR(\vec{v}_5) + \frac{c(1-c)}{4} \vec{v}_2 \\ &+ \frac{c(1-c)}{4} \vec{v}_4 \end{aligned}$$

Then, compute  $PPR(\vec{v}_1)$  with equation solver using the initial guess  $PPR_0(\vec{v}_1)$

**Figure 4 Initial Guess Revision for Two Cases**

decomposition theorem is not applicable. In this case, by the initial guess construction algorithm presented above,  $PPR(\vec{v}_2)$  and  $PPR(\vec{v}_4)$  are replaced by  $c\vec{v}_2$  and  $c\vec{v}_4$ . The resulting guess is as follows.

$$\begin{aligned} PPR_0(\vec{v}_1) &= c\vec{v}_1 + \frac{1-c}{4} PPR(\vec{v}_3) + \frac{1-c}{4} PPR(\vec{v}_5) + \frac{c(1-c)}{4} \vec{v}_2 \\ &+ \frac{c(1-c)}{4} \vec{v}_4 \end{aligned}$$

Then we can compute  $PPR(\vec{v}_1)$  by performing SOR iteration starting from the initial guess  $PPR_0(\vec{v}_1)$ .

In summary, we construct initial guess by using  $c\overrightarrow{v_{neid}}$  instead of  $PPR(\overrightarrow{v_{neid}})$  when  $PPR(\overrightarrow{v_{neid}})$  is unknown.  $c\overrightarrow{v_{neid}}$  is the first term of the decomposition theorem equation:  $PPR(\overrightarrow{v_{neid}}) = c\overrightarrow{v_{neid}} + \frac{1-c}{|O(n_{neid})|} \sum_{n_j \in Out(n_{neid})} PPR(\overrightarrow{v_j})$ . The second summation term can be considered as insignificant term since the values in each dimension of  $\frac{(1-c)^2}{|O(n_{neid})|} PPR(\overrightarrow{v_j})$  becomes negligible value by repetitive multiplication of small scalar values. The performance gain from using the incomplete guess is empirically evaluated in chapter 4.3.

We can inspect more than two-hop neighbors to find known PPR vectors instead of checking only the direct neighbor nodes. In this way we can utilize more information to construct initial guesses. However, the process can break simplicity of the original algorithm and one important merit of our method can be lost as a result. Keeping simplicity of process can contribute to overall performance by making it easier to parallelize. Adding recursive search routine makes the algorithm less uniform consequently makes it harder to parallelize. Such situation should be avoided in massive data processing since parallelization is one of the most important point in optimizing large scale systems.

In addition, if we extend the searching process infinitely, the algorithm becomes the same as the bookmark coloring algorithm (BCA) we discussed in chapter 2.4.3. It is just an algorithmic variant of the plain power iteration and suffers from slow convergence rate.

The above algorithm constructs the initial guesses for  $PPR(\overrightarrow{v_i})$ , that is, PPR vectors for one hot vector queries. However, a query vector  $\vec{q}$  of PPR can be any stochastic vector. In this case, we cannot use the algorithm we discussed before. Instead, we can use *linearity* we discussed in chapter 2.2 to handle the general query vectors. By



the property,  $PPR(\vec{q})$  for any stochastic vector  $\vec{q}$  can be obtained by weighted summation of  $PPR(\vec{v}_i)$ . For example, if we want to compute  $PPR(\vec{q})$  where  $\vec{q}[i] = 0.3$ ,  $\vec{q}[j] = 0.7$ , and all other entries are 0, then, by the linearity property of PPR,  $PPR(\vec{q})$  can be obtained by the following equation.

$$PPR(\vec{q}) = 0.3 * PPR(\vec{v}_i) + 0.7 * PPR(\vec{v}_j)$$

We can use the property to construct initial guess in the same way as decomposition theorem: we utilize the known vectors directly, and use estimation  $c\vec{v}_i$  for the unknown vectors  $PPR(\vec{v}_i)$ . However, the quality of the initial guesses constructed from the method can be low if the weights corresponding to the unknown vectors are dominant comparing to the weights for the known vectors. To avoid the situation, computing at least  $PPR(\vec{v}_i)$  in the precomputation phase is desirable to handle general PPR queries. The detail about the issue is discussed in the following chapter 4 and 5.

In this chapter, we explained how the iterative method for PPR computation can be accelerated by initial guess revision and over relaxation. One may question that the two strategies are generic methods that can be applied to any problem other than PPR computation. However, our algorithm for finding optimal weight for successive over relaxation is a unique method designed for PPR computation, and the initial guess construction algorithm is enabled by the mathematical properties of PPR. Thus, we can claim that our contribution is more than just applying generic methods to PPR problem. We designed the optimal PPR computation algorithm considering the

characteristics of PPR.

In the following chapters, we discuss how the concepts and strategies introduced in this chapter can be applied to the FPPR problem and the PPR query processing and contribute to boost the processes significantly.

## Chapter 4

### Fully Personalized PageRank Algorithm with Initial Guess Revision

In this chapter, we propose an algorithm for fully personalized PageRank (FPPR) problem that computes  $PPR(v_i)$  for each node  $n_i$  in the given graph. As we discussed in the previous chapter, we can boost PPR computation process if we have already computed PPR vectors. How the core idea of initial guess revision can contribute to optimization of FPPR is explained throughout the following sections.

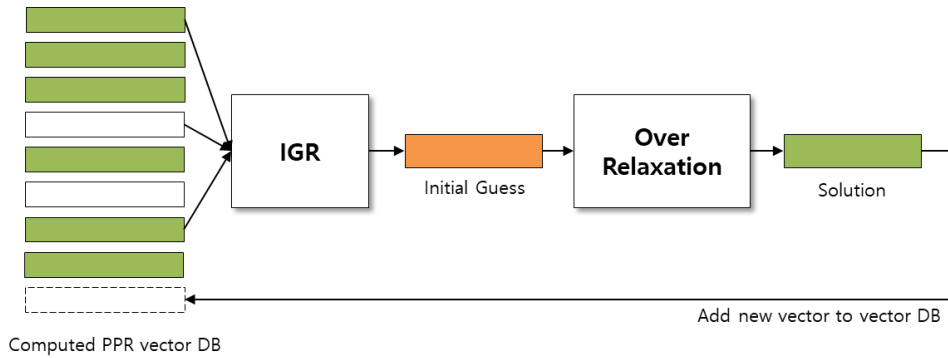
#### 4.1. FPPR with IGR

The basic idea of our FPPR algorithm with IGR is to reuse the PPR vectors that are computed and stored in the previous process. As we stated before, FPPR process produces  $PPR(v_i)$  for each node  $n_i$ . Assuming that each  $PPR(v_i)$  is computed sequentially, the resulting vector DB is gradually filled with PPR vectors one by one during FPPR process. It means that the information that can be utilized by IGR algorithm is populating throughout FPPR process. Thus, we can use them for constructing initial guess to reduce total running time of the FPPR process.

The overall algorithm is described as follows:

- Sort nodes by PageRank
- For each node  $n_i$  in  $G$ 
  - Compute  $PPR(v_i)$  by the following process
    1. Construct initial guess by IGR scheme
    2. Run iteration with over relaxation

First, all nodes are sorted by PageRank value. In this way,  $PPR(\vec{v}_l)$  for relatively more influential nodes are acquired in early stage, and IGR can be applied to more cases. Then, for each node  $n_i$ , initial guess  $PPR_0(\vec{v}_l)$  is computed by IGR algorithm, and iterative process boosted by over relaxation finds the solution  $PPR(\vec{v}_l)$ . In this way, we can efficiently compute  $PPR(\vec{v}_l)$  for each node  $n_i$  in the given graph.



**Figure 5 FPPR with IGR Overview**

Figure 5 illustrates how FPPR with IGR works. To compute  $PPR(\vec{v}_i)$ , the initial guess for  $PPR(\vec{v}_i)$  is constructed first by referencing already computed PPR vectors. The algorithm that we discussed in the chapter 3.2 is used in this stage. Then over relaxation process computes  $PPR(\vec{v}_i)$  efficiently by using the initial guess as its starting point. Once the  $PPR(\vec{v}_i)$  is computed, the vector is feedbacked to the vector DB and can be referenced in initial guess revision process for other nodes. Thus, more vectors become available to be referenced with the FPPR process progresses.

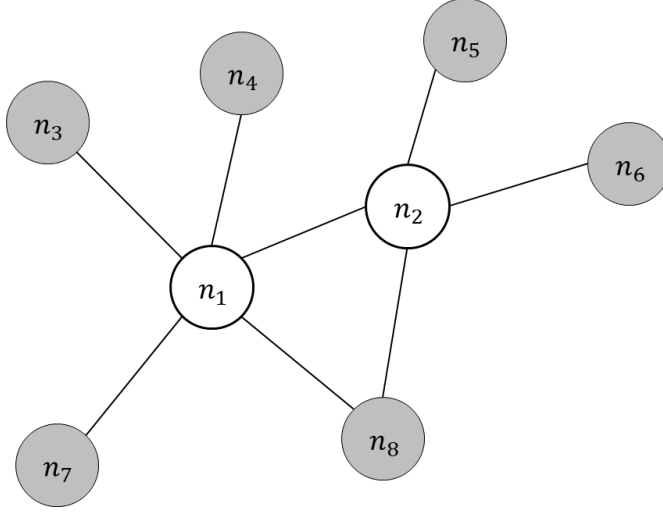
As a result, at the latter part of the FPPR process, most PPR vectors can be directly computed by decomposition theorem without iterative updates. If keeping strict error bound is required, iterative updates should be done even when PPR vector for every neighbor node is known since error can be propagates with summation of decomposition theorem. However, even in the case, additional iteration count needed is dramatically reduced by initial guess revision utilizing decomposition theorem since it produces the initial point that is already extremely close to the target point.

On the contrary, in the early stage of the FPPR process, we have only a small number of vectors to reference in vector DB. Thus, the quality of initial guesses that we use in the early stage is inevitably crude. However, it can be compensated by computing PPR vectors for the highly influential nodes first. This is why we sort nodes by PageRank value before starting the main loop of FPPR process. By acquiring and storing the vectors that can be frequently referenced by the IGR process in early stage, the negative effect of cold start can be minimized. We can also use other centrality measures such as degree of each node. However, in most cases, sorting by PageRank is effective since it measures the importance of each node by reflecting overall structure of graphs.

Once the initial guess is determined, then successive over relaxation finds the final solution for the target node. The iteration scheme may not be necessarily the successive over relaxation. Even plain power iteration is also applicable, and it does not negate the performance gain from initial guess revision. However, over relaxation scheme provides highly faster convergence than the plain power iteration as we discussed in the chapter 3.1. The empirical evaluation of the performance gain by adopting successive over relaxation is discussed in the following chapter 4.3. Other linear iterative solvers such as GMRES does not handle large and sparse matrices that PPR computation needs and initial guess revision scheme does not work with those methods. Thus, we can claim that SOR is one of most reasonable choice for the PPR iteration with initial guess revision

We can also easily determine the optimal SOR weight for PPR iteration with simple algebraic operation  $w_{opt} = 1 + (\frac{c}{1+\sqrt{1-c^2}})^2$  as we discussed in the chapter 3.1. The cost for acquiring the  $w_{opt}$  is negligible. It does not involve the analysis on the graph structure such as clustering, and the resulting optimal weight only depends on the damping factor  $c$ . It is important fact since it means that PPR can be easily accelerated by SOR. Even in the case of the assumption that the transition matrix  $P$  is a proper stochastic matrix does not hold, the theoretic optimal weight can be utilized as a good estimation of the real optimal weight. Once the optimal weight is determined, it can be reused throughout whole FPPR process since it does not vary with query vectors.

Let us assume that we compute  $PPR(\vec{v}_1)$  and  $PPR(\vec{v}_2)$  on the example graph in figure 6 sequentially, and the PPR vectors for the other nodes connected to the three nodes are already computed. First, decomposition theorem cannot be applied to



**Figure 6 Example Graph for FPPR**

compute  $PPR(\vec{v}_1)$  since  $PPR(\vec{v}_2)$ , is not stored in the vector DB yet. Instead, the initial guess construction algorithm we discussed in chapter 3.3 is applied here. By the algorithm, the unknown vector  $PPR(\vec{v}_2)$  is replaced by the estimation  $c\vec{v}_2$ . Thus, the initial guess for  $PPR(\vec{v}_1)$  is computed by the following equation.

$$PPR_0(\vec{v}_1) = c\vec{v}_1 + \frac{c(1-c)}{5} v_2 + \frac{1-c}{5} PPR(\vec{v}_3) + \frac{1-c}{5} PPR(\vec{v}_4) \\ + \frac{1-c}{5} PPR(\vec{v}_7) + \frac{1-c}{5} PPR(\vec{v}_8)$$

The denominator 5 comes from the outdegree of  $n_1$ . In this way, we can acquire an estimated vector that are close to the target vector  $PPR(\vec{v}_1)$ . Once the initial guess  $PPR_0(\vec{v}_1)$ , the guess is refined by SOR. The iterative computation is performed until

the predefined precision requirement is satisfied. The precision requirement is generally defined by the vector norms of the vector difference between  $PPR_k(\vec{v}_1)$  and  $PPR_{k+1}(\vec{v}_1)$ . If we use L2 norm, then the iteration is repeated until  $\|PPR_k(\vec{v}_1) - PPR_{k+1}(\vec{v}_1)\|_2$  drops down under the predefined error bound. Once the stop condition is satisfied, the resulting vector  $PPR_{k+1}(\vec{v}_1)$  is considered as the target vector  $PPR(\vec{v}_1)$  that we searched for. The next step is, as we discussed before, to store the resulting vector  $PPR(\vec{v}_1)$  in the vector DB. Then, we can move on to the next vector  $PPR(\vec{v}_2)$ .

The situation is different for  $PPR(\vec{v}_2)$ . Since we already computed the  $PPR(\vec{v}_1)$ , every PPR vector for the neighbor of  $n_2$  stored in the vector DB. Thus, we can compute  $PPR(\vec{v}_2)$  by applying decomposition theorem as follows.

$$PPR(\vec{v}_2) = c\vec{v}_2 + \frac{1-c}{4} PPR(\vec{v}_1) + \frac{1-c}{4} PPR(\vec{v}_5) + \frac{1-c}{4} PPR(\vec{v}_6) + \frac{1-c}{4} PPR(\vec{v}_8)$$

In this case, the iterative computation is not required. However, as we discussed before, additional iterations should be performed if keeping strict precision requirement is important. If the strictness is not important, to store the PPR vector without further iteration is recommended since the quality gain from the refinement process is not significant.

As we can see in the example process, the nodes that processed later can reference more PPR vectors. In early stage, PPR vectors should be computed nearly from



scratch; however, most PPR vectors computed in latter stage can be derived directly by combining the stored PPR vectors by applying decomposition theorem. This effect is well shown by the experiment using Enron e-mail dataset presented in the chapter 4.3. The required iterations count for most nodes go down to nearly zero after the one fourth of PPR vectors are computed.

After repeating the process until the  $PPR(\vec{v}_i)$  for each node  $n_i$  is computed and stored in the vector DB, the algorithm ends. It produces PPR vectors for one hot vectors as its result. However, with the vector DB, we can compute  $PPR(\vec{q})$  for any stochastic vector  $\vec{q}$ . The linearity property of PPR explained in chapter 2.2. By the property,  $PPR(\vec{q})$  for any stochastic vector  $\vec{q}$  can be obtained by weighted summation of  $PPR(\vec{v}_i)$ . For example, if we want to know  $PPR((0.2, 0, 0, 0.8, 0)^T)$ , then we can compute the PPR vector by the following equation:  $PPR((0.2, 0, 0, 0.8, 0)^T) = 0.2 * PPR(\vec{v}_0) + 0.8 * PPR(\vec{v}_4)$ . However, the error propagation issue also should be considered here. Since  $PPR(\vec{v}_i)$  is not exact solution with zero error, the error within the stored PPR vectors can propagate to the result. To prevent the situation that the resulting error gets larger than the predefined precision requirements, the same solution can be applied as other cases; we can refine the solution by iterations. However, in most practical situations, the vector produced by the linearity property is good enough if the errors within the stored vectors are below predefined error bound.

By the linearity property of PPR, we can even compute PPR vectors for non-stochastic query vectors such as  $PPR(\vec{1})$ . However, they are not meaningful information for graph analysis.

The FPPR algorithm discussed in this chapter assumes that there is no precomputed

vector before the algorithm is performed. However, in the practical situation, the batch process to compute PPR vectors is periodically performed on the same graphs database with some updates, and in most cases only small portion of the graph is updated. It means that the precomputed vectors are available, and they can be used as good initial guesses. The major merit of using iterative method comes from the situation. Iterative method can reach the solution quickly by using the precomputed solutions as its initial guesses while other methods should recompute the solution from zero ground. The performance gain from using precomputed solutions is significant. Further discussions and experiments on this issue are provided in the chapter 5 with explanations on PPR query processing algorithm.

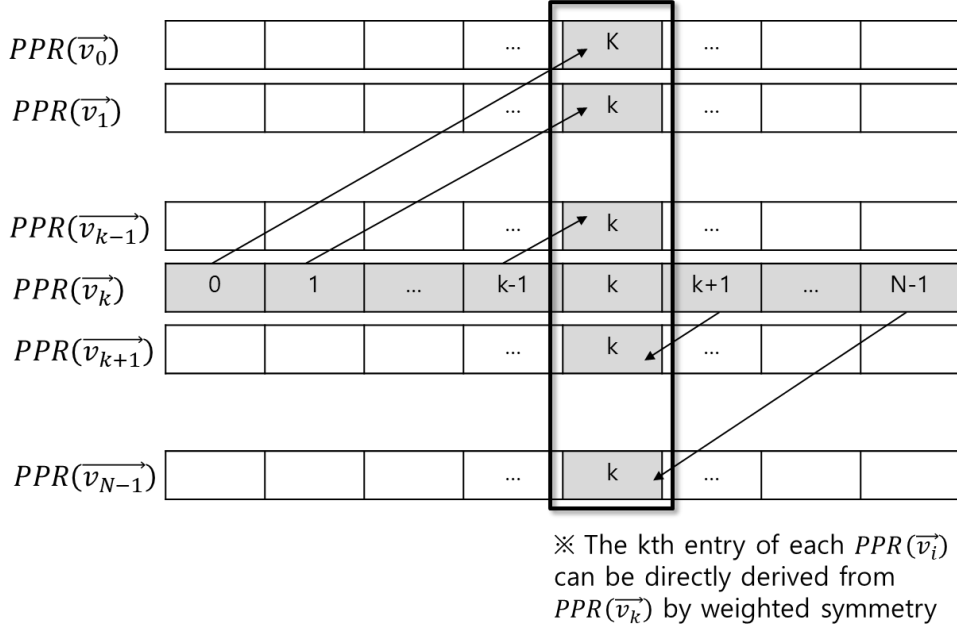
## 4.2. Optimization

The FPPR process can be further optimized utilizing one of mathematical characteristics of PPR, the *weighted symmetry* explained in the chapter 2.2. If the graph is undirected graph, we can derive the value of certain dimensions of PPR vectors without performing iteration by the following formula.

$$PPR(\vec{v}_i)[j] = \frac{degree(j)}{degree(i)} PPR(\vec{v}_j)[i]$$

It means that we acquire  $PPR(\vec{v}_i)[j]$  for every node  $i$ , if  $PPR(\vec{v}_j)$  is already computed and stored in vector DB. The figure 6 illustrates how the optimization scheme works. Once we know  $PPR(\vec{v}_k)$ ,  $i$ th entry of  $PPR(\vec{v}_k)$  can be utilized to

determine  $PPR(\vec{v}_i)[k]$ . In other words, if the  $k$ th row of the vector DB is already filled, the  $k$ th column also can be decided with little cost.



**Figure 7 Optimization Scheme Utilizing Weighted Symmetry Property**

This property enables powerful optimization for FPPR. By utilizing the property, we can decide half of total entries in resulting vector DB without iterative computation. Thus, it nearly halves total computation time. We also can say that the dimension that iteration method handles diminishes throughout whole FPPR process. If we have  $M$  computed PPR vectors, the next PPR vector can be determined by computing the values for  $N - M$  dimensions since the values for  $M$  dimensions are already known. After computing the PPR vector, the dimension count to be decided to compute the next PPR vector decreases to  $N - M - 1$ .

However, the optimization scheme is only effective when whole resulting vector DB fits in main memory since it requires repetitive random access to the individual entries in the vectors. Though the basic FPPR process also requires random access operations to reference the computed PPR vectors, the optimized computation utilizing the weighted symmetry requires large numbers of fine level random accesses toward individual entries across multiple vectors to compute even one PPR vector. It can cause critical performance degradation when the computed PPR vectors are stored in disks.

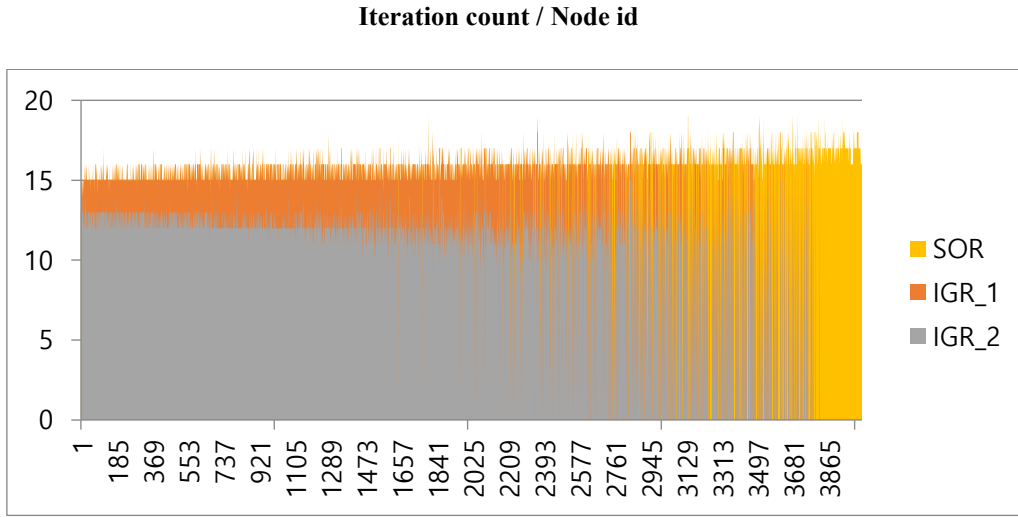
In summary, this additional optimization technique has two major restrictions: the graph should be an undirected graph, and the resulting vector DB ( $O(N^2)$ ) should fit in main memory. Thus, this optimization technique is applicable to only limited situations. However, once the above conditions are satisfied, it can dramatically accelerate whole FPPR process by cutting down the half of total computation time. Considering the fact that the size of main memory is continuously increasing, we can claim that it is also one of important optimization techniques that should not be overlooked.

The fact that division and multiplication can enlarge errors also should be noted. Even  $PPR(\vec{v}_j)$  is the result of iterations with certain error criteria,  $\frac{degree(j)}{degree(i)} PPR(\vec{v}_j)$  can have larger error than the criteria. Therefore, if the strict error bound should be kept, additional iterations should be performed after setting  $PPR(\vec{v}_j)[j]$  as  $\frac{degree(j)}{degree(i)} PPR(\vec{v}_j)$ . However, since the value  $PPR(\vec{v}_j)[j]$  is already extremely close to the solution, only small number of iterations are needed to adjust the result. In most practical situation, to use the solution without further iteration is recommended considering the trade-off between computing time and

quality of results. Generally, the quality gain from the additional iteration does not worth starting new iteration process.

### 4.3. Experiments

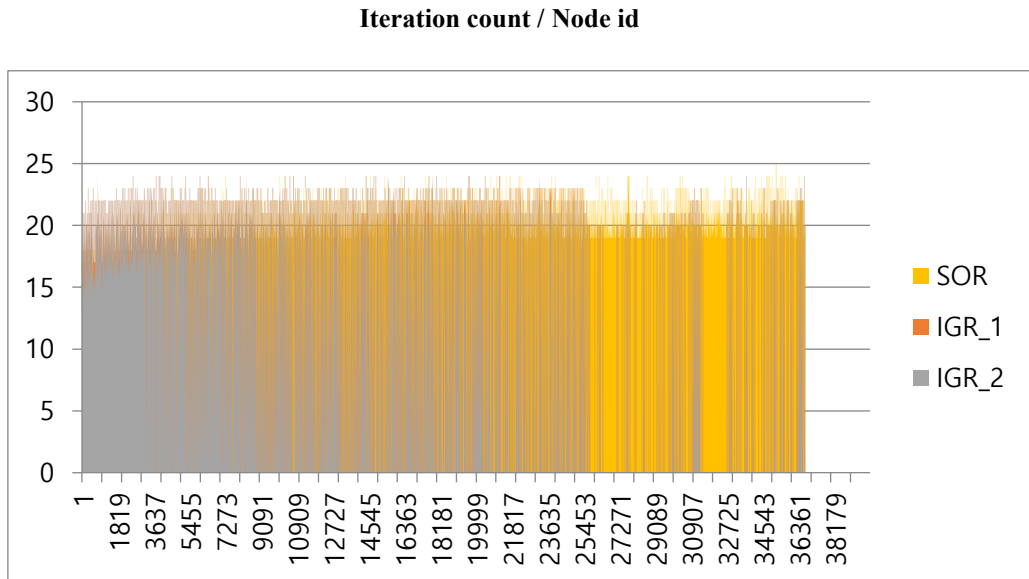
In this section we provide the empirical evaluations for FPPR algorithm using initial guess revision. The performance gain from adopting each strategy (initial guess revision, successive over relaxation, and optimization using weighted symmetry) is evaluated with experimental results.



The figure 8 show the effect of introducing IGR to FPPR process. We use the Facebook graph data with 4,039 nodes. X-axis indicates the node number, and Y-axis indicates the required iteration count to compute  $PPR(\vec{v}_i)$  for the node  $n_i$ .

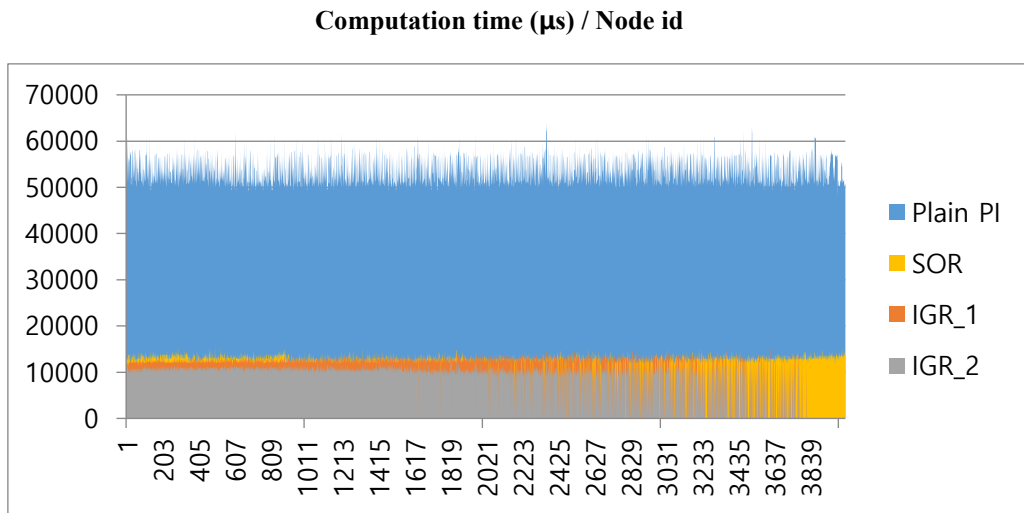
SOR indicates the case that only relaxation scheme is applied, IGR\_1 indicates the case that relaxation and decomposition theorem is applied, and IGR\_2 indicates the case that the full algorithm described above is applied. In early stage, the benefit from IGR is not significant. However, the performance gain expands through the process. As a result, overall iteration count is reduced by 21.3% with IGR\_1 and 38.9% with IGR\_2.

The impact of adopting IGR is more obviously shown with the larger dataset. The following graph is the result of the same experiment with Enron e-mail graph dataset with 36,692 nodes. In this case, the effect of using IGR is shown in earlier stage, and overall iteration count is reduced by 72.7%.



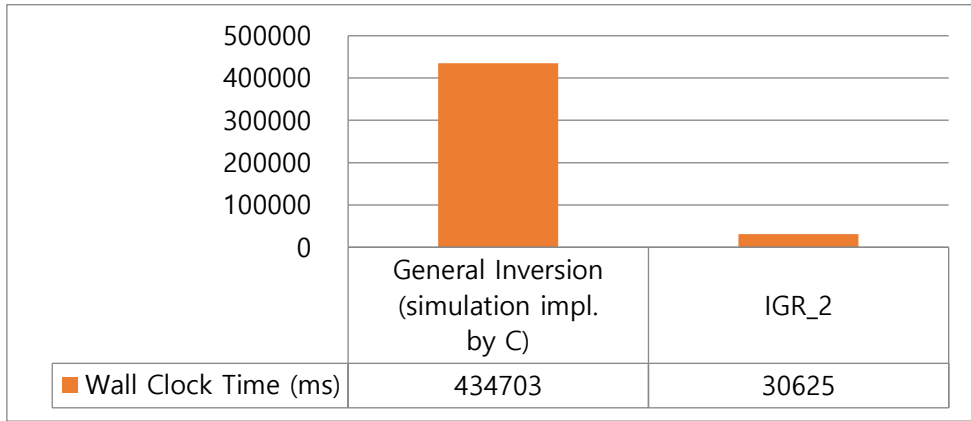
**Figure 9 Iteration Count per Node (Enron E-mail Graph)**

The following graph compares plain power iteration and our method with the Facebook graph data. X-axis indicates the node number, and Y-axis indicates the required time in microseconds to compute  $PPR(\vec{v}_i)$  for the node  $n_i$ . It shows that introducing relaxation scheme largely reduces running time. Overall running time is reduced by 84.1%. With the Enron e-mail graph dataset, overall running time is reduced by 90.3%



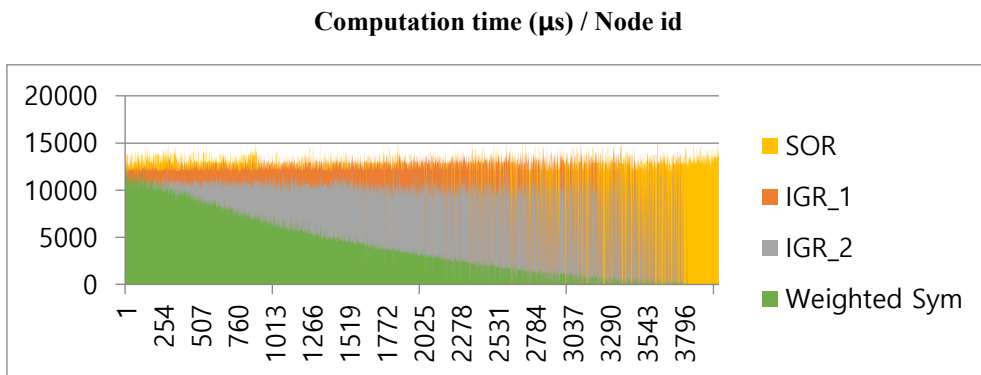
**Figure 10 Computation Time per Node (Facebook Graph)**

The following graph compares matrix inversion and our method. We compare our method and matrix inversion with generic LU decomposition. The result shows that our method is significantly faster than LU decomposition; our algorithm is 14 times faster than the LU decomposition.



**Figure 11 Comparison with General Matrix Inversion**

The figure 12 shows the effect of optimization strategy utilizing weighted symmetry property of PPR. It is obvious that it nearly halves total computation times. We can see that the running time for each node drops down by the process progresses since the number of directly derivable dimension is linearly increasing along with the number of computed PPR vectors.



**Figure 12 Effect of Utilizing Weighted Symmetry (Facebook Graph)**



## Chapter 5

### Personalized PageRank Query Processing with Initial Guess Revision

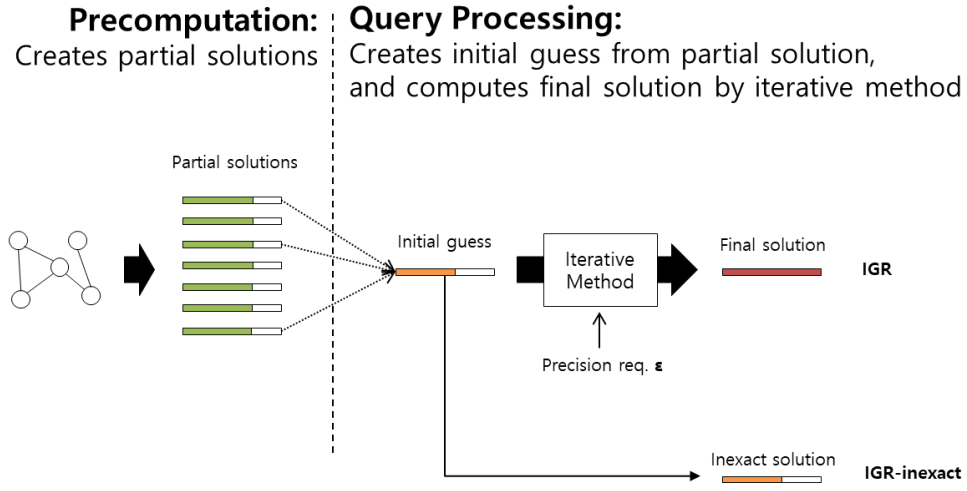
The FPPR algorithm we discussed in Chapter 4 computes and stores  $PPR(v_i)$  for each node  $n_i$  in the given graph. However, it is not always applicable since it requires  $O(N^2)$  space to store all resulting PPR vectors. It is obviously impractical to store all vectors if  $N$  is over 1 million. Thus, to handle very large graphs, we need to compute  $PPR(v_i)$  on query time utilizing well-designed precomputed data structure, and it also can be improved with IGR and over relaxation scheme.

As we stated in the chapter 2, high precomputation cost is one of common problem of recently proposed PPR computation methods. We explain how our method, IGR with over relaxation, solves the issue throughout chapter 5.

#### 5.1. PPR Query Processing with IGR

The figure 13 illustrates how PPR query processing algorithm works with IGR scheme. Query processing consists of two stages; the Precomputation stage where

partial solutions are created before runtime, and the Query Processing stage where actual results are computed utilizing the partial solutions.



**Figure 13 Query Processing with Initial Guess Revision**

First, the partial solutions are computed and stored in precomputation phase. This process is similar to FPPR algorithm; however, in this case, only top-k truncated vectors are stored while FPPR stores full vectors. That is, only top-k largest dimensions in the resulting PPR vectors are stored. For example, if the resulting full vector is  $(0.25, 0.5, 0.2, 0.05, 0)^T$ , top-2 truncated vector stores the following information:  $(0.25, 0.5, 0, 0, 0)^T$ . Though it has the same number of values as a full vector, it can be stored more efficiently since it has high sparsity.

Other parts of the algorithm work exactly the same as the FPPR version. First, the nodes are sorted by their PageRank, and a top-k truncated vector for each node is computed one by one with IGR and over relaxation. However, in this case, the quality

of initial guesses is inevitably degraded since only top- $k$  dimensions are available for initial guess construction and the effect of IGR scheme can be reduced as a result. In other words, speed is sacrificed to reduce storage cost. However, in most cases, weight distribution within PPR vectors is highly skewed. A few dimensions monopolize most weight while others have nearly zero weight. Thus, the strategy storing and referencing only top- $k$  dimensions can provide balanced performance considering both time and space. The empirical evaluation results provided in the following chapter 5.2 prove that the strategy can reduce large portion of space cost while paying little cost in terms of speed.

Storing strategy can be changed. That is, truncation can be done by proportion instead of dimension count. We can truncate out the entries with values smaller than predefined threshold. However, under those strategies, the length of storing vectors can varies. It can make the overall process less uniform thus cause performance degradation when massive data processing is required.

Keeping full vectors instead of truncated ones for the selected set of influential nodes can be beneficial. However, for massive graphs with over 1 million nodes, the cost for storing even one full vector is not trivial. If  $N = 1,000,000$  and  $k = 100$ , storing one full vector requires the same space for 10K truncated vectors. Thus, it is hard to say that the strategy is efficient for massive graph handling.

The algorithm explained above assumes that no PPR vectors to reference are available at the beginning of whole process. However, in most practical situation, we have more information. It is the result of former precomputation result. The precomputation phase should be performed periodically to provide query results reflecting the recent state of the graph database. Thus, the precomputation to handle

the updates within graphs can utilize the PPR vectors produced by the former precomputation to construct initial guesses. The PPR vectors cannot be considered as proper PPR vectors for any node for the updated graph. Once the graph is updated, the PPR vectors computed from the previous graph get outdated. However, if we assume that the update was not revolutionary one that change nearly half of nodes, we can say that the PPR vector is still close to the real PPR vectors we try to compute. It means that the “outdated” PPR vectors themselves can be considered as high-quality initial guesses. Since they are already close to the target point, the updated PPR vectors can be computed with small number of iterations.

This strategy dramatically reduces the cost of update handling. Most recently proposed methods that require structural analysis of the given graphs in their precomputation phase cannot reuse the result of the former precomputation, and they should repeat the costly precomputation phase from scratch. On the other hand, our method solves the issue by utilizing the self-repairing characteristic of iterative method. The impact of reusing the previous precomputed data is well shown by the experiments presented in chapter 5.3.

In query time, we have two options. One is to compute the full PPR vectors for the query vectors using the initial guesses constructed from the precomputed top-k truncated vectors, and the other one is to return the initial guesses themselves as answers for the queries.

The initial guess for  $PPR(\vec{q})$  can be constructed in multiple ways. If the query vector  $\vec{q}$  is a one hot vector  $\vec{v}_i$ , one of the stored top-k truncated vectors can be used as an initial guess. However, better initial guesses can be constructed by combining the PPR vectors of the neighbor nodes of  $n_i$ . If the query vector is not a

one hot vector, the initial guess for  $PPR(\vec{q})$  can be constructed by utilizing the *linearity* property of PPR explained in chapter 2.2. By the property,  $PPR(\vec{q})$  for any stochastic vector  $\vec{q}$  can be obtained by weighted summation of  $PPR(\vec{v}_i)$ . Since the top-k truncated vector of  $PPR(\vec{v}_i)$  for each node  $n_i$  is computed in precomputation phased, we can estimate  $PPR(\vec{q})$  with the truncated vectors.

For example, in the example graph in figure 6, the initial guess for the  $PPR(\vec{v}_1)$  can be constructed by combining the precomputed top-k truncated vectors with decomposition theorem as follows:

$$\begin{aligned} PPR_0(\vec{v}_1) = c\vec{v}_1 + \frac{1-c}{5} \widehat{PPR(\vec{v}_2)} + \frac{1-c}{5} \widehat{PPR(\vec{v}_3)} + \frac{1-c}{5} \widehat{PPR(\vec{v}_4)} \\ + \frac{1-c}{5} \widehat{PPR(\vec{v}_7)} + \frac{1-c}{5} \widehat{PPR(\vec{v}_8)} \end{aligned}$$

$\widehat{PPR(\vec{v}_i)}$  denotes the top-k truncated vector of  $PPR(\vec{v}_i)$ . In this case, there is no unknown case since all  $\widehat{PPR(\vec{v}_i)}$  are computed in the precomputed phase. One thing should be noted is that it is not the exact solution even though there is no unknown neighbor. Since  $PPR_0(\vec{v}_1)$  is summation of partial information, the result is also just an estimation. The iteration should be performed to find the target  $PPR(\vec{v}_i)$ .

Since  $\widehat{PPR(\vec{v}_1)}$  is also stored in the vector DB, we can use it as an initial guess. However, the above guess constructed by decomposition theorem can provide less sparse initial guess, and the cost of the vector summation process is nearly negligible

with modern parallel computing capability. Thus, it is reasonable to use the  $PPR_0(\vec{v}_1)$  as an initial guess instead of  $\widehat{PPR}(\vec{v}_1)$ .

The initial guesses for general vectors other than one hot vector also can be done in the nearly the same way as the FPPR case. The only one thing is different. The top-k truncated vectors are used in this case. For example, if we want to know  $PPR((0.2, 0, 0, 0.8, 0)^T)$ , then we can construct the initial guess by the following equation:

$$PPR_0((0.2, 0, 0, 0.8, 0)^T) = 0.2 * \widehat{PPR}(\vec{v}_0) + 0.8 * \widehat{PPR}(\vec{v}_4)$$

As we stated before, the fact that the result is not the PPR vector should be noted. The vector is just a combination of partial vectors and it should not be considered as a PPR vector.

We can consider the two-level initial guess construction scheme for general queries. For example,  $PPR_0((0.2, 0, 0, 0.8, 0)^T) = 0.2 * PPR_0(\vec{v}_0) + 0.8 * PPR_0(\vec{v}_4)$  can be used instead of  $PPR_0((0.2, 0, 0, 0.8, 0)^T) = 0.2 * \widehat{PPR}(\vec{v}_0) + 0.8 * \widehat{PPR}(\vec{v}_4)$ . This algorithm can produce definitely better initial guesses. However, it requires multiple initial guess construction process as its subprocess, and it should be repeated as the number of non-zero dimensions within the query vector. If the query vector has large number of non-zero dimensions, the cost for constructing initial guess can be larger than the iterative process itself. The situation should be avoided.

The query processing algorithm we discussed above returns the exact PPR vectors through the refinement process. However, if query answering time is more important

than the quality of query answers, we can use the second option (i.e. **IGR-Inexact**). As we stated before, weight distribution of PPR vectors is highly skewed, and several top dimensions monopolize most weights. Thus, we can consider the initial guesses derived from top-k truncated vectors as high-quality estimations for the final solutions even though they are not PPR vectors. Thus, IGR-Inexact returns  $PPR_0(\vec{q})$  we discussed before as its final result. The quality of the strategy is empirically evaluated in the following chapter 5.3. Though the strategy provides good balance between quality and speed, it sacrifices several useful features of IGR scheme since it does not perform the refinement process in query time. The discussion on the issue is provided in chapter 5.2 along optimization tips.

The following pseudo code summarizes the overall process of PPR query processing algorithm with IGR scheme:

- Precomputation Phase
  - Sort nodes by PageRank
  - Compute top-k truncated vector of each node with IGR and Over Relaxation
  - If we have former precomputation result for the node, use it for IGR and compute top-k truncated vector by Over Relaxation
- Query Processing
  - (IGR) Compute  $PPR(\vec{q})$  with initial guess produced by decomposition theorem and linearity

- For one-hot vector queries, construct initial guesses by combining top-k truncated vectors by decomposition theorem
  - For general queries, construct initial guesses by combining top-k truncated vectors by linearity
- (IGR-Inexact) Return the initial guess as an answer for the given query without iteration

Like the FPPR algorithm, successive over relaxation is also used for finding the target PPR vectors from initial guesses. The strategy for determining the optimal weight for SOR iteration is also the same as the FPPR algorithm. Once  $w_{opt}$  is determined in precomputation phase, we do not need to reselect  $w_{opt}$  on query time since it does not depend on query vectors. The theoretic optimal weight  $w_{opt} = 1 + (\frac{c}{1+\sqrt{1-c^2}})^2$  still can be utilized for query processing.

Another notable feature of the IGR based PPR query processing method is that it can answer the exact solution for the updated graph even without redoing precomputation. Though the top-k truncated vectors produced from the previous precomputation get outdated once the graph is updated, as we stated before, they still can be considered as high-quality initial guesses. With the initial guesses, the iterative method (SOR in our method) can reach the exact solution by its self-repairing capability.

However, it does not mean the precomputation phase is not required after very first precomputation. After the updates accumulate within the graph, the precomputed



top-k truncated vectors cannot produce high quality initial guesses since it does not reflect the updates. As a result, query processing speed possibly slows down. It means that the effect and purpose of the precomputation phase of IGR is different from other PPR computation method. Generally, the precomputation phase is required for keeping correctness of the query answers; however, in our method, the precomputation phase is required for maintaining query processing performance. In other methods, the query results get outdated without precomputations. However, our method can avoid the situation by utilizing self-repairing property of iterative method though the query answering speed slows down in some degree. This property of our method can be helpful when robustness is important. One thing should be noted is that the property is not valid in IGR-Inexact since IGR-Inexact does not perform iteration in query time.

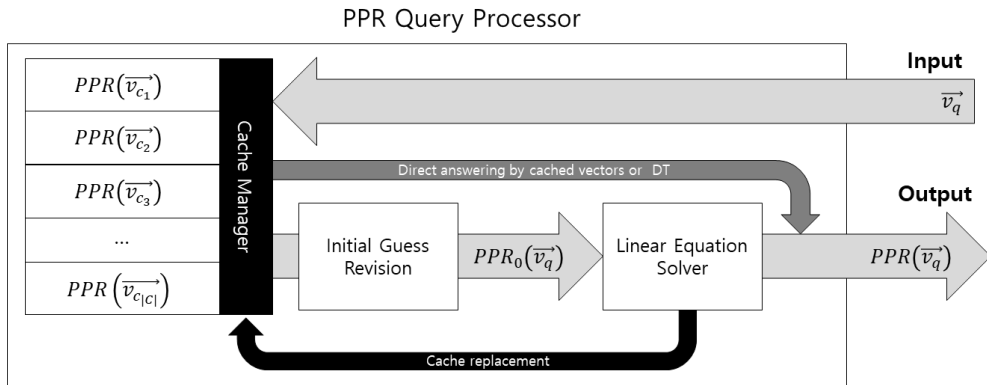
## 5.2. Optimization

For the queries that request PPR vector for single query nodes (i.e.  $PPR(\vec{v}_l)$ ), The answers can be feedbacked to precomputed data structure since the answers are the PPR vectors for the updated graph data. With IGR scheme, the correct  $PPR(\vec{v}_l)$  is always produced by self-repairing capability of iterative method. Thus, once  $PPR(\vec{v}_l)$  is computed on query time, we do not need to compute it again in the next precomputation phase since we already know the correct PPR vector for the node.

With the live update strategy, the cost for periodic precomputation can be further optimized by avoiding redundant computation. If we have lively updated top-k truncated vectors, the iteration count we need in the precomputation phase can be dramatically reduced. Though the strategy is not applicable if PPR for single nodes

are not requested in query time, in most practical applications, it can provide powerful additional optimization on the top of basic IGR scheme. This scheme also is not valid in IGR-Inexact since IGR-Inexact does not perform refinement process on query time.

Maintaining full vector caches for frequently referenced PPR vectors can be another idea for further optimization. However, as we stated before, the additional space cost for storing full vector is remarkably high. If we handle the graph with 1 million nodes, it requires multiple megabytes to store one full PPR vector. Maintaining those full vector caches on the main memory can burdens overall system. Thus, one should carefully consider the trade-off between additional space cost and performance gain by paying the cost. The figure 14 illustrates an example of PPR query processing system design utilizing caching scheme. PPR query processing can be optimized by caching full vectors on a selective basis utilizing various replacement policies.



**Figure 14 Example System Design Utilizing Caching Scheme**

Iterative process in the precomputation phase also can be further optimized. Since the result of precomputation is top-k truncated vector DB, stop conditions other than vector norms can be used. In the FPPR problem, the PPR vectors from the batch computation are the final product of the process. Thus, they should be computed by keeping the accuracy requirements. However, in this case, the PPR vectors from the batch computation (that is, precomputation phase) are not the final outputs of the system. They are just ingredients for constructing initial guesses. Thus, in this case, stop condition can be lessened. In other word, we can stop the iteration earlier.

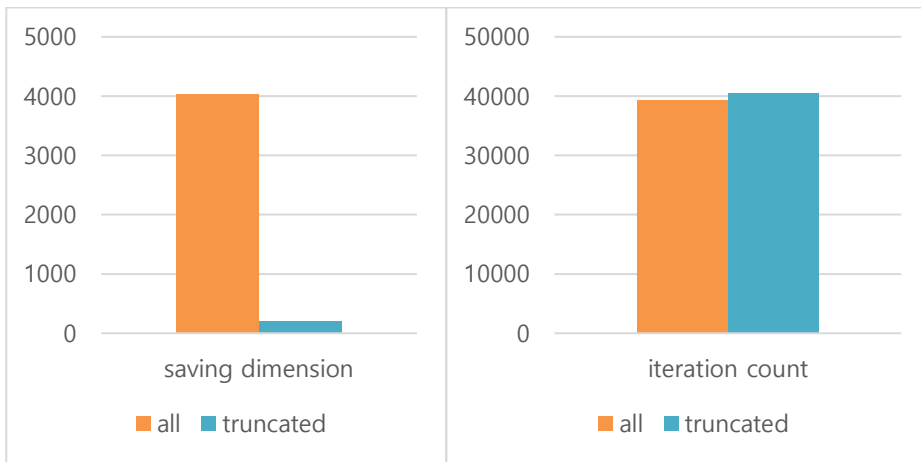
For example, top-k order can be an option. Iteration can stop when the order of top-k dimension does not change. With this condition, iterations can stop much earlier than the FPPR iterations. The resulting vectors are pseudo top-k truncated vectors. However, they are enough for constructing good initial guesses. However, checking top-k order for each iteration is needed to utilize the stop condition. The cost for sorting dimensions can be problematic since sorting process requires super linear time. Repeating the costly process on each iteration step possibly degrades overall performance of the precomputation phase. Thus, we can use the following alternative strategy: first, for small set of nodes, compute pseudo top-k truncated vectors by iterating until top-k order does not change and store error when iteration ends. Then, for other nodes, iterate until error is reduced below the stored error bound. In this way, we can avoid sorting all dimensions on every iteration.

These optimization tips do not necessarily improve performance. Under some circumstances, they possibly degrade the performance. However, they can provide meaningful alternative options when additional improvement is required in the diverse practical situations.

### 5.3. Experiments

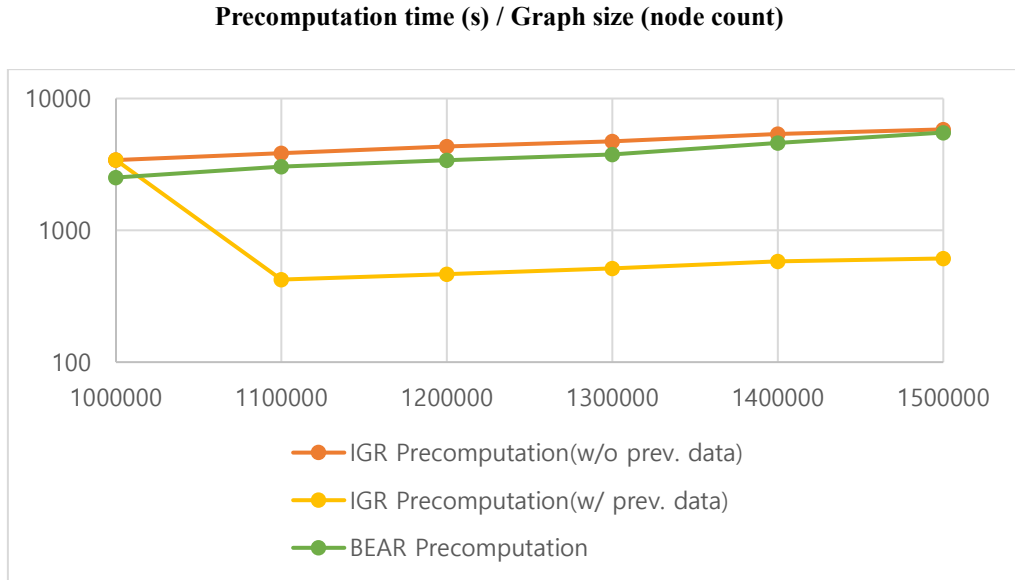
In this section we provide the empirical evaluations for PPR query processing algorithm using initial guess revision. The performance gain from adopting each strategy (storing top-k truncated vectors, reusing former precomputation results, and query processing with IGR-Inexact) is evaluated with experimental results.

The figure 15 shows that the saving dimensions and total iteration count of precomputation phase. “all” indicates the case that we store full vectors (4,039 dimensions) and use them in IGR, and “truncated” indicates the case that we store only top-200 dimensions and use them in IGR. We use the Facebook graph data with 4,039 nodes. The implication of the result is obvious. We can reduce 95.0% of space cost with 2.9% of time loss. Significant amount of storage cost is reduced by paying nearly negligible cost in terms of time. If we do not have surprisingly cheap storage and we do not have to sacrifice everything for optimizing processing time, it is reasonable to save only top-k truncated vectors instead of full vectors.

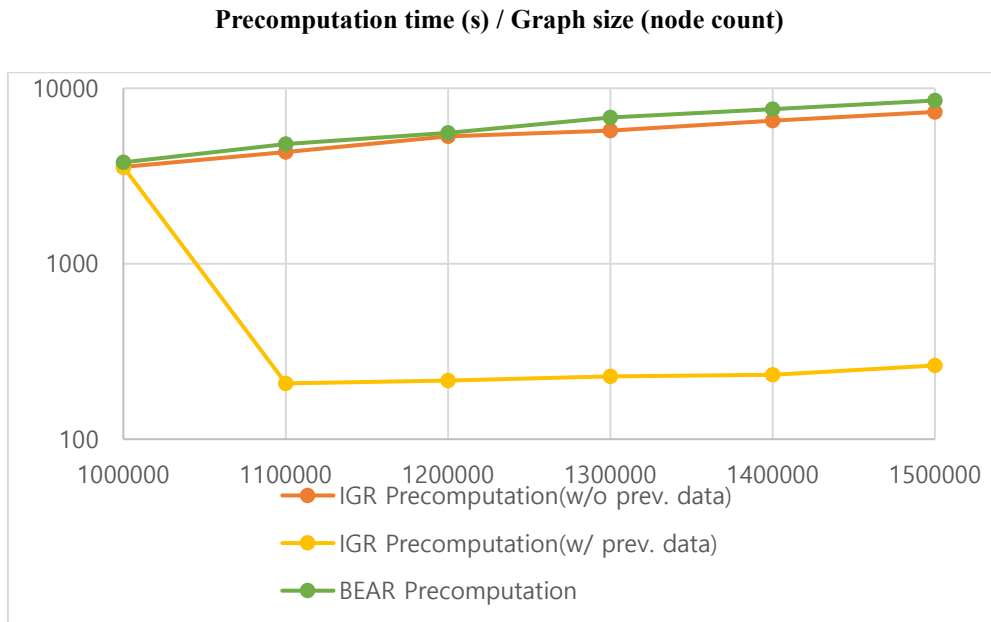


**Figure 15 Effect of Using Truncated Vectors**

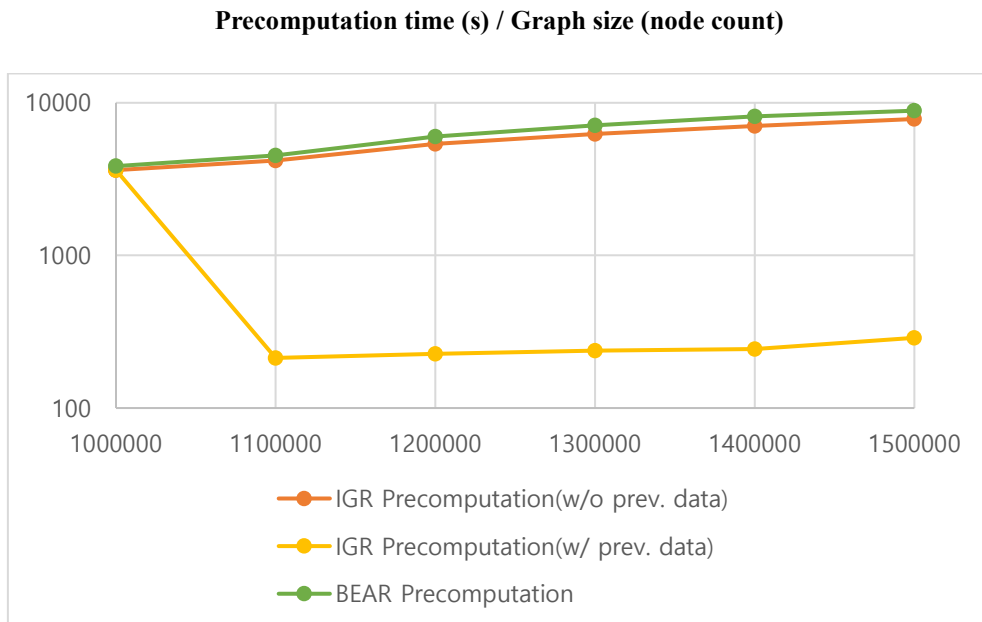
The figure 16 ~ 18 show the performance gain from using the vectors produced by former precomputation phase. For simulating updates, we use synthesized using GraphStream library. We generate three types of graphs: scale-free network, random graph, and small world graph. We simulate graph updates by adding 100K new nodes to the original graph with 1M nodes on each step. New edges also are added according to each graph model. Then, we measure the performance gain from the vector reusing strategy by comparing the precomputation performance with/without the previous precomputation data for each updated graph. We also compare the performance of our algorithm with the precomputation algorithm of state-of-the-art PPR computation algorithm BEAR. X-axis indicates the size of the graph in node count, and Y-axis indicates running time in seconds.



**Figure 16 Effect of Re-using Previous Precomputed Data (Scale-Free Network)**



**Figure 17 Effect of Re-using Previous Precomputed Data (Random Graph)**



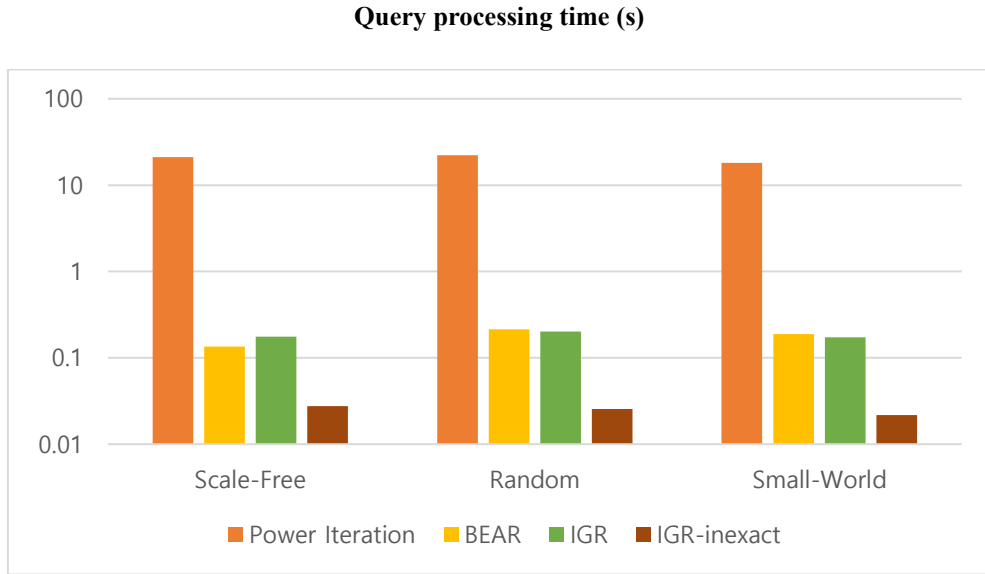
**Figure 18 Effect of Re-using Previous Precomputed Data (Small World Graph)**

Without utilizing the former precomputation data, IGR is slower than BEAR on scale-free network. However, reusing previous result significantly reduces running time, and it outperforms the state-of-the-art algorithm by order of magnitude. It means that our strategy of recycling “outdated” vectors as initial guesses was highly effective. Though our method performs slower in the cold start case, considering the practical situation with periodic updates, it is important to handling updates since cold start case occurs only once while periodic updates should be handled multiple times. We also can say that our strategy highly enhances the capability of dynamic graph handling. Considering that large number of graphs analyzed by PPR such as World Wide Web and social networks have high dynamicity, we can say that our method provides an effective solution one of core issues in PPR computation.

This improvement is achieved by introducing iterative method to PPR computation and enhancing it. As we stated before, iterative method has self-repairing capability. With wrong or imperfect solutions are given, iterative method can fix the solution by iterative process, and it can find the solution faster with better initial guesses while other recent algorithms such as matrix decomposition-based methods cannot utilize imperfect solutions.

The graph model does not affect overall performance. however, IGR performs slightly better when the graph is not scale-free network. It is because that BEAR is clustering-based algorithm. In scale-free network the presence of node clusters is more obvious thus BEAR can utilize it. However, in the general cases that node clusters are not relatively obvious, IGR performs faster than BEAR. We can say that IGR gets fewer negative effects from structural characteristics of the graphs while BEAR slows down when graph does not have certain features.

The figure 19 shows query processing performance on the synthesized graph with 1M nodes generated by GraphStream library. IGR is faster than plain power iteration by order of magnitude, and it shows comparable performance to the state-of-the-art algorithm, BEAR. Though the experiment report that IGR is slower than BEAR, IGR can answer query faster if accuracy is sacrificed since the stored top-k truncated vectors and the initial guesses generated from the vectors can be considered as an inexact but meaningful answer. Considering that the quality evaluation result for the IGR-Inexact provided in the latter part of this chapter, it is reasonable to use IGR-Inexact if only top-k node lists are needed for the answers.



**Figure 19 Query Processing Time**

The graph model affects the query processing performance in similar way as the precomputation performance. Though it is hard to say that the model has significant



effect on the query processing performance, BEAR slows down when the graph is not scale-free network while the performance of IGR is relatively consistent.

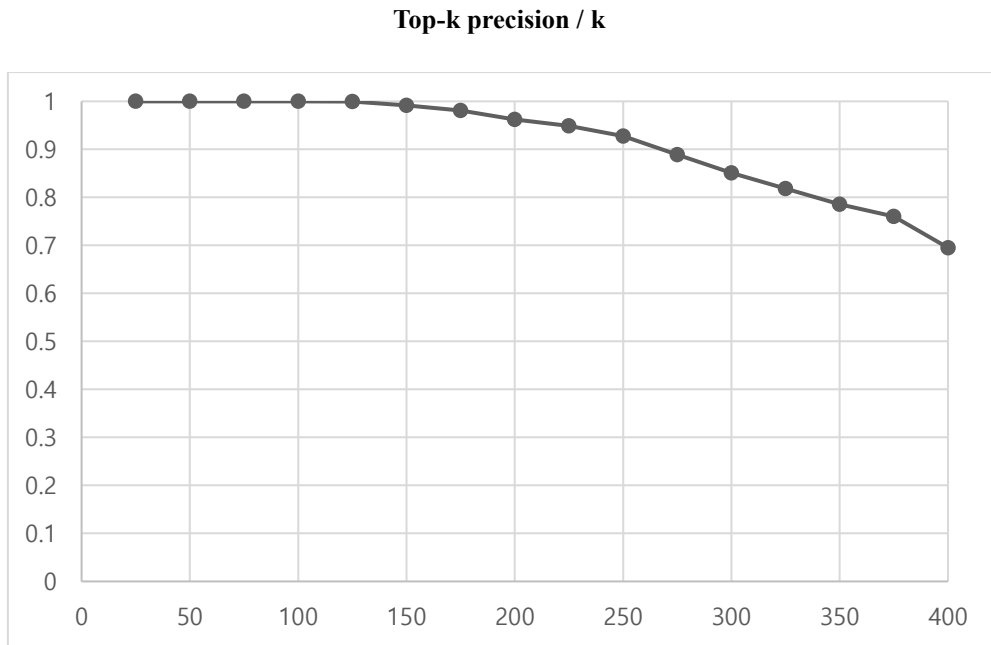
The following table shows the quality of inexact solution of PPR query processing algorithm by comparing cosine similarity. We use top-100 truncated vectors. In the case of one-hot vector queries that requests  $PPR(\vec{v}_i)$ , the similarity score is near 1.00 since most weights in the top-k dimensions. However, in the case of general vector queries, the similarity score is lower since it requires reference more top-k truncated vectors and the results reflect more errors. However, it still over 0.90 and we can say that the quality result can be acquired by efficiently with inexact method.

**Table 5 Accuracy of Result in Cosine Similarity**

	One-hot vector queries	General vector queries
Power Iteration	1.0000	1.0000
BEAR	1.0000	1.0000
IGR	1.0000	1.0000
<b>IGR-inexact</b>	<b>0.9873</b>	<b>0.9523</b>

The figure 20 shows top-k precision of IGR-Inexact using top-100 truncated vectors. The result shows that initial guesses obtained by combining top-100 truncated vectors also can be useful query answer. It shows over 90% top-k precision for top-200 queries, and 85% for top-300 queries. Though only top-100 dimensions are stored for each node, IGR process combines multiple top-100 truncated vectors and more than 100 dimensions can be returned for the answers of top-k queries. However, the coverage is clearly limited since IGR-Inexact utilizes limited source data. If one needs accurate top-k results for high k value, iterative process needs to be done in

query time. Saving more dimensions in precomputation phase also can be an alternative solution.



**Figure 20 Top-k Precision of IGR Inexact (Using Top-100 Truncated Vectors)**

## Chapter 6

### Conclusion

In this thesis, efficient algorithm for computing personalized PageRank (PPR) on large graphs is proposed. Unlike the recent studies on the topic, we have introduced an enhanced iterative scheme that overcomes the limitations of previous methods.

Most previous methods cannot handle update efficiently, and they rerun costly precomputation phase again for every update. However, our method fixes the former precomputation results by utilizing the characteristics of iterative scheme and constructing better initial guess from the results with the IGR algorithm. We also further boost the iterative process with over relaxation scheme. By applying over relaxation, iteration is performed efficiently while preserving simplicity of power iteration algorithm that is suitable for large scale parallel implementation.

Based on the idea, we have proposed two algorithms: FPPR algorithm, and PPR query processing algorithm. FPPR algorithm computes and stores  $PPR(v_i)$  vector for each node  $n_i$  in the given graph, and PPR query processing algorithm computes and stores top-k truncated  $PPR(v_i)$  vector for each node  $n_i$  in the given graph and uses them to boost query processing. Both algorithms can handle updates on graphs with surprising efficiency by reusing the vectors generated from the former precomputation phase. This is the main contribution of this thesis, and the

achievement comes from utilizing the self-repairing characteristic of iterative methods.

Our algorithm can be applied to relationship analysis on large dynamic graphs like World Wide Web. Unlike previous algorithms that recompute all PPR vectors each time from scratch, with our method, the update process can be performed more efficiently based on former results, and precomputation time can be dramatically shortened. Thus, refresh process can be done more frequently, and the analytic system can reflect more recent states of dynamic graphs with dramatically enhanced efficiency.

Graph-based recommendation is also a good example to which our algorithm can be applied to. If PPR is used for generating recommendation results based on the data modelled as a large graph, the PPR vectors should be updated periodically (e.g. once a day) to provide updated recommendation to customers reflecting recent activities. If updating PPR vectors cannot be performed efficiently and it takes impractically long time (e.g. more than a day to refresh entire vectors), the capability of recommendation can be severely limited. Our algorithm can solve the situation by greatly enhanced update handling capability. Except for cold start cases, our update scheme based on vector reusing strategy accelerates the update handling performance by order of magnitude. Efficient update handling can contribute to reducing the refreshing cycle of PPR vectors, and to realizing recommendation services with enhanced liveliness. We expect that our method facilitates introduction of graph-based recommendation algorithms to the industry by removing one of most important barriers for using them in practical situations.

The idea of initial guess revision can be applied to extended problems. Though it cannot be applied to the recently proposed iterative methods such as GMRES,

simpler methods such as fixed-point iteration are still important to the domains that can be modeled as large and sparse matrices. For fixed-point iteration methods, initial guess revision scheme can be a powerful solution as we discussed throughout the thesis.

The parallelization of our method can be the next step of our study. Since large part of the algorithm consists of matrix-vector multiplication, it can be easily parallelized. Data splitting scheme to achieve optimal performance for the IGR algorithm also can be a meaningful research problem.

## Bibliography

- [1] H. Tong, C. Faloutsos, and J.-Y. Pan, “Fast random walk with restart and its applications,” in Sixth International Conference on Data Mining (ICDM’06), pp. 613–622, IEEE, 2006.
- [2] D. Fogaras, B. R’acz, K. Csalog’any, and T. Sarl’os, “Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments,” *Internet Mathematics*, vol. 2, no. 3, pp. 333–358, 2005.
- [3] D. Fogaras and B. R’acz, “Towards scaling fully personalized pagerank,” in *International Workshop on Algorithms and Models for the Web-Graph*, pp. 105–117, Springer, 2004.
- [4] B. Bahmani, K. Chakrabarti, and D. Xin, “Fast personalized pagerank on mapreduce,” in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pp. 973–984, ACM, 2011.
- [5] S. Chakrabarti, A. Pathak, and M. Gupta, “Index design and query processing for graph conductance search,” *The VLDB Journal*, vol. 20, no. 3, pp. 445–470, 2011.
- [6] P. Berkhin, “Bookmark-coloring algorithm for personalized pagerank computing,” *Internet Math.*, vol. 3, no. 1, pp. 41–62, 2006.
- [7] T. H. Haveliwala, “Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search,” *IEEE transactions on knowledge and data engineering*, vol. 15, no. 4, pp. 784–796, 2003.
- [8] L. Sun, C. Cheng, X. Li, D. Cheung, and J. Han, “On link-based similarity join,” *Proceedings of the VLDB Endowment*, 2011.

- [9] Y. Fujiwara, M. Nakatsuji, M. Onizuka, and M. Kitsuregawa, “Fast and exact top-k search for random walk with restart,” *Proceedings of the VLDB Endowment*, vol. 5, no. 5, pp. 442–453, 2012.
- [10] T. Maehara, T. Akiba, Y. Iwata, and K.-i. Kawarabayashi, “Computing personalized pagerank quickly by exploiting graph structures,” *Proceedings of the VLDB Endowment*, vol. 7, no. 12, pp. 1023–1034, 2014.
- [11] P. A. Lofgren, S. Banerjee, A. Goel, and C. Seshadhri, “Fast-ppr: Scaling personalized pagerank estimation for large graphs,” in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1436–1445, ACM, 2014.
- [12] T. Sarló, A. A. Benczúr, K. Csalogány, D. Fogaras, and B. Rácz, “To randomize or not to randomize: space optimal summaries for hyperlink analysis,” in *Proceedings of the 15th international conference on World Wide Web*, pp. 297–306, ACM, 2006.
- [13] Y. Z. Guo, K. Ramamohanarao, and L. A. Park, “Personalized pagerank for web page prediction based on access time-length and frequency,” in *IEEE/WIC/ACM International Conference on Web Intelligence (WI’07)*, pp. 687–690, IEEE, 2007.
- [14] F. Pedroche, F. Moreno, A. González, and A. Valencia, “Leadership groups on social network sites based on personalized pagerank,” *Mathematical and Computer Modelling*, vol. 57, no. 7-8, pp. 1891–1896, 2013.
- [15] E. Agirre and A. Soroa, “Personalizing pagerank for word sense disambiguation,” in *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 33–41, Association for Computational Linguistics, 2009.

- [16] Y. Liu, X. Wang, J. Zhang, and H. Xu, "Personalized pagerank based multidocument summarization," in IEEE International Workshop on Semantic Computing and Systems, pp. 169–173, IEEE, 2008.
- [17] H.-N. Kim and A. El Saddik, "Personalized pagerank vectors for tag recommendations: inside folkrank," in Proceedings of the fifth ACM conference on Recommender systems, pp. 45–52, ACM, 2011.
- [18] S. Lee, S. Park, M. Kahng, and S.-g. Lee, "Pathrank: a novel node ranking measure on a heterogeneous graph for recommender systems," in Proceedings of the 21st ACM international conference on Information and knowledge management, pp. 1637–1641, ACM, 2012.
- [19] T. H. Kim, K. M. Lee, and S. U. Lee, "Generative image segmentation using random walks with restart," in European conference on computer vision, pp. 264–275, Springer, 2008. 2
- [20] B. Ham, D. Min, and K. Sohn, "A generalized random walk with restart and its application in depth up-sampling and interactive segmentation," IEEE Transactions on Image Processing, vol. 22, no. 7, pp. 2574–2588, 2013.
- [21] G. Iv'an and V. Grolmusz, "When the web meets the cell: using personalized pagerank for analyzing protein interaction networks," Bioinformatics, vol. 27, no. 3, pp. 405–407, 2010.
- [22] K. Avrachenkov, N. Litvak, D. Nemirovsky, and N. Osipova, "Monte carlo methods in pagerank computation: When one iteration is sufficient," SIAM Journal on Numerical Analysis, vol. 45, no. 2, pp. 890–904, 2007.
- [23] G. Jeh and J. Widom, "Scaling personalized web search," in Proceedings of the 12th international conference on World Wide Web, pp. 271–279, Acm, 2003.



- [24] G. Jeh and J. Widom, “Simrank: a measure of structural-context similarity,” in Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 538–543, ACM, 2002.
- [25] P. Sarkar and A. W. Moore, “Fast nearest-neighbor search in disk-resident graphs,” in Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 513–522, ACM, 2010.
- [26] D. Gleich and M. Mahoney, “Anti-differentiating approximation algorithms: A case study with min-cuts, spectral, and flow,” in International Conference on Machine Learning, pp. 1018–1025, 2014.
- [27] K. Shin, J. Jung, S. Lee, and U. Kang, “Bear: Block elimination approach for random walk with restart on large graphs,” in Proceedings of the 2015 ACM SIGMOD international conference on Management of Data, pp. 1571–1585, ACM, 2015.
- [28] S. Boyd and L. Vandenberghe, Convex optimization. Cambridge university press, 2004.
- [29] F. Zhu, Y. Fang, K. C.-C. Chang, and J. Ying, “Scheduled approximation for personalized pagerank with utility-based hub selection,” The VLDB Journal—The International Journal on Very Large Data Bases, vol. 24, no. 5, pp. 655–679, 2015.
- [30] S. D. Kamvar, T. H. Haveliwala, C. D. Manning, and G. H. Golub, “Extrapolation methods for accelerating pagerank computations,” in Proceedings of the 12th international conference on World Wide Web, pp. 261–270, ACM, 2003.
- [31] A. N. Langville and C. D. Meyer, Google’s PageRank and beyond: The science of search engine rankings. Princeton University Press, 2011.

- [32] L. Page, S. Brin, R. Motwani, and T. Winograd, “PageRank citation ranking: Bringing order to the web”, tech. rep., Stanford InfoLab, 1999.
- [33] Z. Junchao, C. Junjie, J. Song, and R.-X. Zhao, “Monte carlo based personalized pagerank on dynamic networks,” *International Journal of Distributed Sensor Networks*, vol. 9, no. 9, p. 829804, 2013.
- [34] C. Borgs, M. Brautbar, J. Chayes, and S.-H. Teng, “Multiscale matrix sampling and sublinear-time pagerank computation,” *Internet Mathematics*, vol. 10, no. 1-2, pp. 20–48, 2014.
- [35] I. Konstas, V. Stathopoulos, and J. M. Jose, “On social networks and collaborative recommendation,” in *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pp. 195–202, ACM, 2009.
- [36] S. Lee, S.-i. Song, M. Kahng, D. Lee, and S.-g. Lee, “Random walk based entity ranking on graph for multidimensional recommendation,” in *Proceedings of the fifth ACM conference on Recommender systems*, pp. 93–100, ACM, 2011.
- [37] B. Bahmani, A. Chowdhury, and A. Goel, “Fast incremental and personalized pagerank,” *Proceedings of the VLDB Endowment*, vol. 4, no. 3, pp. 173–184, 2010.
- [38] A. Balmin, V. Hristidis, and Y. Papakonstantinou, “Objectrank: Authority-based keyword search in databases,” in *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pp. 564– 575, VLDB Endowment, 2004.
- [39] Y. Fujiwara, M. Nakatsuji, T. Yamamuro, H. Shiokawa, and M. Onizuka, “Efficient personalized pagerank with accuracy assurance,” in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 15–23, ACM, 2012.

- [40] S. Chakrabarti, “Dynamic personalized pagerank in entity-relation graphs,” in Proceedings of the 16th international conference on World Wide Web, pp. 571–580, ACM, 2007.
- [41] S. Al-Saffar and G. Heileman, “Experimental bounds on the usefulness of personalized and topic-sensitive pagerank,” in Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence, pp. 671–675, IEEE Computer Society, 2007.
- [42] W. Y. Wang, K. Mazaitis, and W. W. Cohen, “Programming with personalized pagerank: a locally groundable first-order probabilistic logic,” in Proceedings of the 22nd ACM international conference on Information & Knowledge Management, pp. 2129–2138, ACM, 2013.
- [43] D. Gleich and M. Polito, “Approximating personalized pagerank with minimal use of web graph data,” *Internet Mathematics*, vol. 3, no. 3, pp. 257–294, 2006.
- [44] J. Xia, D. Caragea, and W. H. Hsu, “Bi-relational network analysis using a fast random walk with restart,” in 2009 Ninth IEEE International Conference on Data Mining, pp. 1052–1057, IEEE, 2009. 4
- [45] H. Tong and C. Faloutsos, “Center-piece subgraphs: problem definition and fast solutions,” in Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 404–413, ACM, 2006.
- [46] Z. Yin, M. Gupta, T. Weninger, and J. Han, “Linkrec: a unified framework for link recommendation with user attributes and graph structure,” in Proceedings of the 19th international conference on World wide web, pp. 1211–1212, ACM, 2010.
- [47] K. Macropol, T. Can, and A. K. Singh, “Rrw: repeated random walks on genome-scale protein networks for local cluster discovery,” *BMC bioinformatics*, vol. 10, no. 1, p. 283, 2009.

- [48] X. Wang, N. Gulbahce, and H. Yu, “Network-based methods for human disease gene prediction,” *Briefings in functional genomics*, vol. 10, no. 5, pp. 280–293, 2011.
- [49] W. Yu and X. Lin, “Irwr: incremental random walk with restart,” in *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pp. 1017–1020, ACM, 2013.
- [50] R. Andersen, F. Chung, and K. Lang, “Local partitioning for directed graphs using pagerank,” *Internet Mathematics*, vol. 5, no. 1-2, pp. 3–22, 2008.
- [51] Z. Dou, R. Song, and J.-R. Wen, “A large-scale evaluation and analysis of personalized search strategies,” in *Proceedings of the 16th international conference on World Wide Web*, pp. 581–590, ACM, 2007.
- [52] S. A. Tabrizi, A. Shakery, M. Asadpour, M. Abbasi, and M. A. Tavallaie, “Personalized pagerank clustering: A graph clustering algorithm based on random walks,” *Physica A: Statistical Mechanics and its Applications*, vol. 392, no. 22, pp. 5772–5785, 2013.
- [53] A. Z. Broder, R. Lempel, F. Maghoul, and J. Pedersen, “Efficient pagerank approximation via graph aggregation,” *Information Retrieval*, vol. 9, no. 2, pp. 123–138, 2006.
- [54] F. Zhu, Y. Fang, K. C.-C. Chang, and J. Ying, “Incremental and accuracyaware personalized pagerank through scheduled approximation,” *Proceedings of the VLDB Endowment*, vol. 6, no. 6, pp. 481–492, 2013.
- [55] E. Garcia, F. Pedroche, and M. Romance, “On the localization of the personalized pagerank of complex networks,” *Linear Algebra and its Applications*, vol. 439, no. 3, pp. 640–652, 2013.
- [56] G. M. Del Corso, A. Gulli, and F. Romani, “Fast pagerank computation via a sparse linear system,” *Internet Mathematics*, vol. 2, no. 3, pp. 251–273, 2005.

- [57] A. N. Langville and C. D. Meyer, “Deeper inside pagerank,” *Internet Mathematics*, vol. 1, no. 3, pp. 335–380, 2004. 5
- [58] J. Jung, N. Park, S. Lee, and U. Kang, “Bepi: Fast and memory-efficient method for billion-scale random walk with restart,” in *Proceedings of the 2017 ACM International Conference on Management of Data*, pp. 789–804, ACM, 2017.
- [59] L. Dong, Y. Li, H. Yin, H. Le, and M. Rui, “The algorithm of link prediction on social network,” *Mathematical Problems in Engineering*, vol. 2013, 2013.
- [60] L. Backstrom and J. Leskovec, “Supervised random walks: predicting and recommending links in social networks,” in *Proceedings of the fourth ACM international conference on Web search and data mining*, pp. 635–644, ACM, 2011.
- [61] W. Liu and L. Lü, “Link prediction based on local random walk,” *EPL (Europhysics Letters)*, vol. 89, no. 5, p. 58007, 2010.
- [62] J. Jung, W. Jin, L. Sael, and U. Kang, “Personalized ranking in signed networks using signed random walk with restart,” in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 973–978, IEEE, 2016.
- [63] R. Devooght, A. Mantrach, I. Kivimäki, H. Bersini, A. Jaimes, and M. Saelens, “Random walks based modularity: application to semisupervised learning,” in *Proceedings of the 23rd international conference on World wide web*, pp. 213–224, ACM, 2014.
- [64] M. Pershina, Y. He, and R. Grishman, “Personalized page rank for named entity disambiguation,” in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 238–243, 2015.

- [65] C. Wang, F. Jing, L. Zhang, and H.-J. Zhang, “Image annotation refinement using random walk with restarts,” in Proceedings of the 14th ACM international conference on Multimedia, pp. 647–650, ACM, 2006.
- [66] J.-S. Kim, J.-Y. Sim, and C.-S. Kim, “Multiscale saliency detection using random walk with restart,” IEEE transactions on circuits and systems for video technology, vol. 24, no. 2, pp. 198–210, 2013.
- [67] H. Kim, Y. Kim, J.-Y. Sim, and C.-S. Kim, “Spatiotemporal saliency detection for video sequences based on random walk with restart,” IEEE Transactions on Image Processing, vol. 24, no. 8, pp. 2552–2564, 2015.
- [68] S. Lee, J. H. Lee, J. Lim, and I. H. Suh, “Robust stereo matching using adaptive random walk with restart algorithm,” Image and Vision Computing, vol. 37, pp. 1–11, 2015.
- [69] T. H. Kim, K. M. Lee, and S. U. Lee, “Edge-preserving colorization using data-driven random walks with restart,” in 2009 16th IEEE international conference on image processing (ICIP), pp. 1661–1664, IEEE, 2009.
- [70] C. Oh, B. Ham, and K. Sohn, “Probabilistic correspondence matching using random walk with restart,” in BMVC, pp. 1–10, 2012. 6
- [71] J. Sun, H. Shi, Z. Wang, C. Zhang, L. Liu, L. Wang, W. He, D. Hao, S. Liu, and M. Zhou, “Inferring novel lncrna–disease associations based on a random walk model of a lncrna functional similarity network,” Molecular BioSystems, vol. 10, no. 8, pp. 2074–2081, 2014.
- [72] X. Chen, Z.-H. You, G.-Y. Yan, and D.-W. Gong, “Irwrla: improved random walk with restart for lncrna-disease association prediction,” Oncotarget, vol. 7, no. 36, p. 57919, 2016.

- [73] X. Chen, M.-X. Liu, and G.-Y. Yan, “Drug–target interaction prediction by random walk on the heterogeneous network,” *Molecular BioSystems*, vol. 8, no. 7, pp. 1970–1978, 2012.
- [74] J. Li, L. Chen, S. Wang, Y. Zhang, X. Kong, T. Huang, and Y.-D. Cai, “A computational method using the random walk with restart algorithm for identifying novel epigenetic factors,” *Molecular genetics and genomics*, vol. 293, no. 1, pp. 293–301, 2018.
- [75] C. Blatti and S. Sinha, “Characterizing gene sets using discriminative random walks with restart on heterogeneous biological networks,” *Bioinformatics*, vol. 32, no. 14, pp. 2167–2175, 2016.
- [76] K. C. Chipman and A. K. Singh, “Predicting genetic interactions with random walks on biological networks,” *BMC bioinformatics*, vol. 10, no. 1, p. 17, 2009.
- [77] M. Nykl, M. Campr, and K. Jeřek, “Author ranking based on personalized pagerank,” *Journal of Informetrics*, vol. 9, no. 4, pp. 777–799, 2015.
- [78] H. Avron and L. Horesh, “Community detection using time-dependent personalized pagerank,” in *International Conference on Machine Learning*, pp. 1795–1803, 2015.
- [79] Piegorsch and G. E. Casella. Inverting a sum of matrices. In *SIAM Review*, 1990.
- [80] Lofgren, Peter, Siddhartha Banerjee, and Ashish Goel. “Personalized pagerank estimation and search: A bidirectional approach.” *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. pp. 163-172, 2016..
- [81] Wang, S., Tang, Y., Xiao, X., Yang, Y., and Li, Z. “HubPPR: effective indexing for approximate personalized pagerank.” *Proceedings of the VLDB Endowment*, vol. 10, no. 3 pp. 205-216, 2016

- [82] Wang, S., Yang, R., Xiao, X., Wei, Z., and Yang, Y., “Fora: Simple and effective approximate single-source personalized pagerank.” In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 505-514, 2017.
- [83] Yu, W., and Lin, X., “IRWR: incremental random walk with restart.” In Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval, pp. 1017-1020, 2013.
- [84] Zhang, H., Lofgren, P., and Goel, A., “Approximate personalized pagerank on dynamic graphs.” In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1315-1324, 2016.
- [85] Yoon, M., Jin, W., and Kang, U., “Fast and accurate random walk with restart on dynamic graphs with guarantees.” In Proceedings of the 2018 World Wide Web Conference, pp. 409-418, 2018.
- [86] Ohsaka, N., Maehara, T., and Kawarabayashi, K. I., “Efficient pagerank tracking in evolving networks.” In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 875-884, 2015.
- [87] Adsuara, J. E., Cordero-Carrión, I., Cerdá-Durán, P., and Aloy, M. A., “Scheduled relaxation Jacobi method: improvements and applications.” Journal of Computational Physics, Vol. 321, pp. 369-413, 2016.
- [88] Liu, Q., Li, Z., Lui, J., and Cheng, J., “Powerwalk: Scalable personalized pagerank via random walks with vertex-centric decomposition.” In Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, pp. 195-204, 2016.



## Chapter A

## Appendix

**Table 6 Wall Clock Time for FPPR Process (Facebook Graph)**

	Plain PI	SOR	IGR_1	IGR_2	General Inversion
Wall Clock Time (ms)	192812	47125	37172	<b>30625</b>	434703

**Table 7 Query Processing Time (s) for 1M Node Graphs**

	Scale-Free Network	Random	Small-World
Power Iteration	21.1456	22.1631	18.1433
BEAR	0.1344	0.2131	0.1879
IGR	0.1761	0.2011	0.1733
IGR-inexact	<b>0.0277</b>	<b>0.0255</b>	<b>0.0217</b>

**Table 8 Precomputation Time (s) for Scale-Free Network**

Node Count	IGR Precomputation (w/o prev. data)	IGR Precomputation (w/ prev. data)	BEAR Precomputation
1000000	3412	3412	<b>2514</b>
1100000	3838	<b>423</b>	3041
1200000	4317	<b>466</b>	3393
1300000	4735	<b>513</b>	3771
1400000	5385	<b>583</b>	4601
1500000	5831	<b>613</b>	5515

**Table 9 Precomputation Time (s) for Random Graph**

Node Count	IGR Precomputation (w/o prev. data)	IGR Precomputation (w/ prev. data)	BEAR Precomputation
1000000	<b>3562</b>	<b>3562</b>	3784
1100000	4338	<b>208</b>	4801
1200000	5317	<b>216</b>	5578
1300000	5735	<b>228</b>	6823
1400000	6535	<b>233</b>	7601
1500000	7331	<b>263</b>	8515

**Table 10 Precomputation Time (s) for Small World Graph**

Node Count	IGR Precomputation (w/o prev. data)	IGR Precomputation (w/ prev. data)	BEAR Precomputation
1000000	<b>3622</b>	<b>3622</b>	3846
1100000	4188	<b>213</b>	4511
1200000	5367	<b>226</b>	6021
1300000	6235	<b>238</b>	7123
1400000	7035	<b>243</b>	8141
1500000	7831	<b>288</b>	8876

## 초 록

그래프 내에서 개인화된 페이지랭크 (Personalized PageRank, PPR)를 계산하는 것은 검색, 추천, 지식발견 등 여러 분야에서 광범위하게 활용되는 중요한 작업이다. 개인화된 페이지랭크를 계산하는 것은 고비용의 과정이 필요하므로, 개인화된 페이지랭크를 계산하는 효율적이고 혁신적인 방법들이 다수 개발되어왔다. 그러나 수백만 이상의 노드를 가진 대용량 그래프에 대한 효율적인 계산은 여전히 해결되지 않은 문제이다. 그에 더하여, 기존 제시된 알고리즘들은 그래프 갱신을 효율적으로 다루지 못하여 동적으로 변화하는 그래프를 다루는 데에 한계점이 크다. 본 연구에서는 높은 정밀도를 보장하고 정밀도를 통제 가능한, 빠르게 수렴하는 개인화된 페이지랭크 계산 알고리즘을 제시한다. 전통적인 거듭제곱법 (Power Iteration)에, 축차가속완화법 (Successive Over Relaxation)과 초기 추측값 보정법 (Initial Guess Revision)을 활용한 벡터 재사용 전략을 적용하여, 수렴 속도를 개선하였다. 제시된 방법은 기존 거듭제곱법의 장점인 단순성과 엄밀성을 유지하면서도 수렴율과 계산속도를 크게 개선한다. 또한 개인화된 페이지랭크 벡터의 갱신을 위하여 이전에 계산되어 저장된 벡터를 재사용하여, 갱신에 드는 시간이 크게 단축된다. 본 방법은 주어진 오차 한계에 도달하는 즉시 결과값을 산출하므로 정확도와 계산시간을 유연하게 조절할 수 있으며, 이는 표본

기반 추정방법이나 정확한 값을 산출하는 역행렬 기반 방법이 가지지 못한 특성이다. 실험 결과, 본 방법은 거듭제곱법에 비하여 20배 이상 빠르게 수렴한다는 것이 확인되었으며, 기 제시된 최고 성능의 알고리즘보다 우수한 성능을 보이는 것 또한 확인되었다.

**주요어** : 그래프 분석, 페이지랭크, 개인화된 페이지랭크, 랜덤 워크, 거듭제곱법, 최적화, 알고리즘

**학번** : 2009-30917